Introducing F#

Don Syme
MSR Cambridge

Introducing F#

Don Syme
MSR Cambridge

Outline

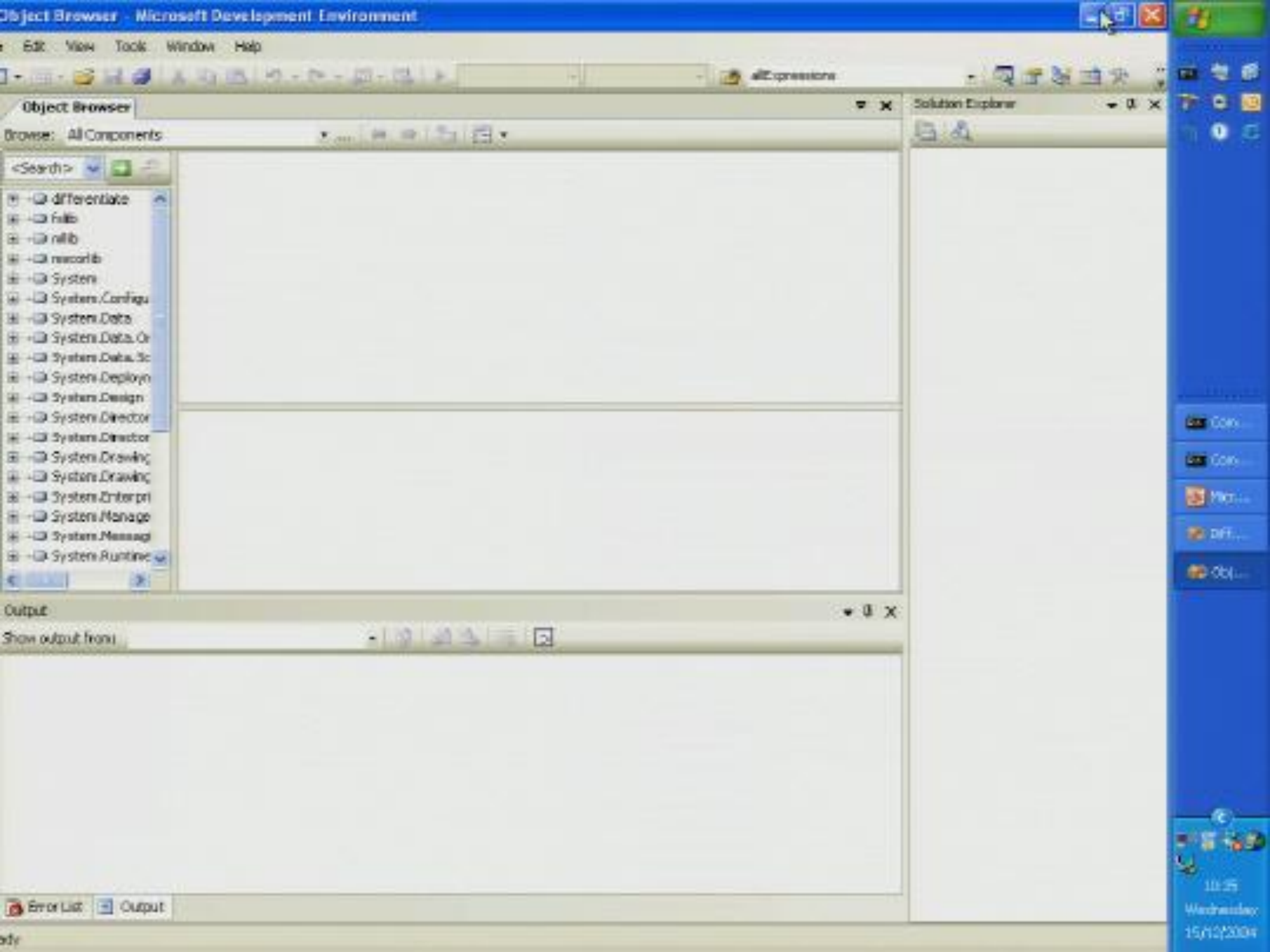
- F#: a quick orientation via a demo
- F#: orthogonal, simple features
- F#: co-operates in the context of .NET

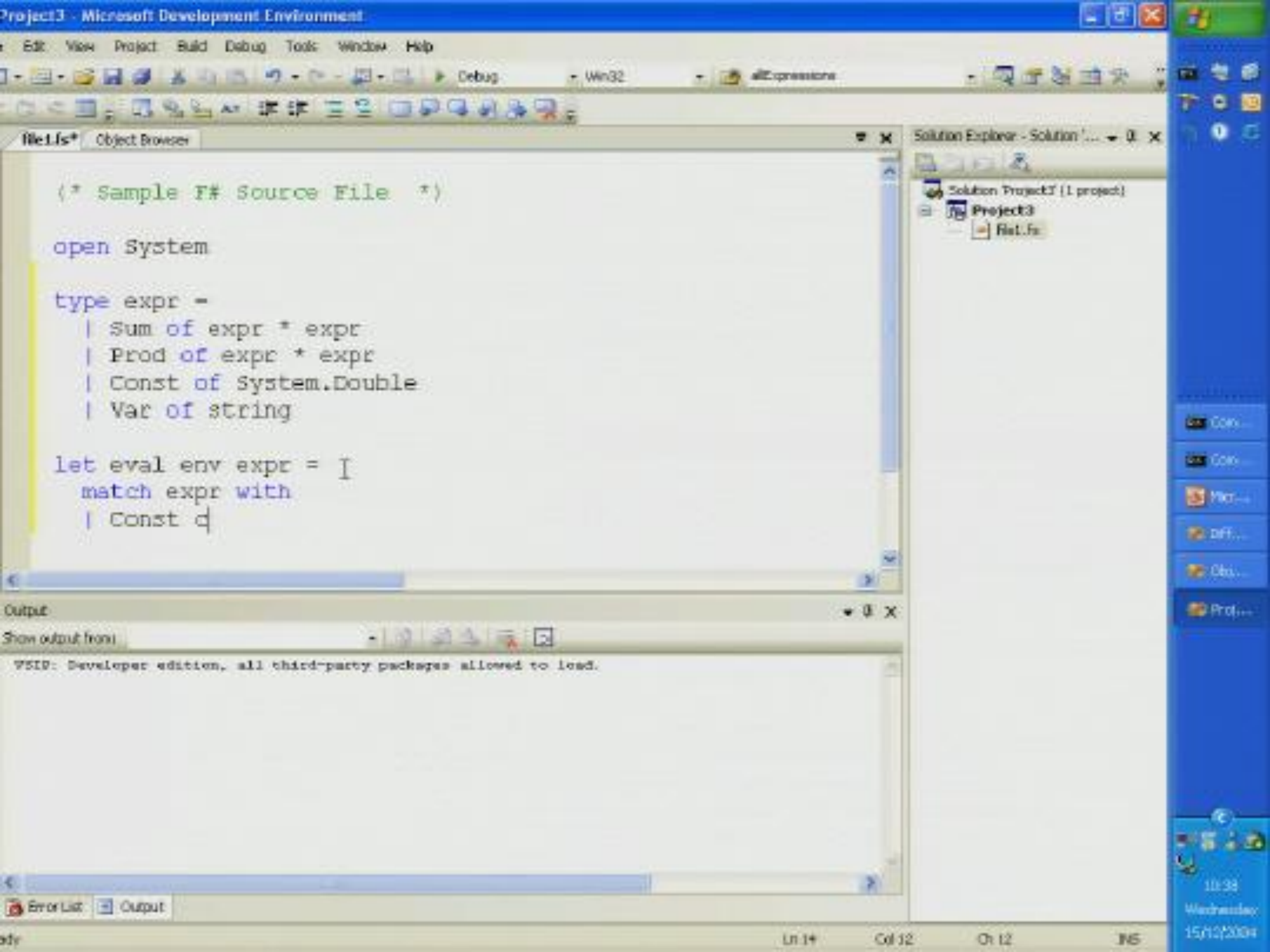
An ML-F# Orientation

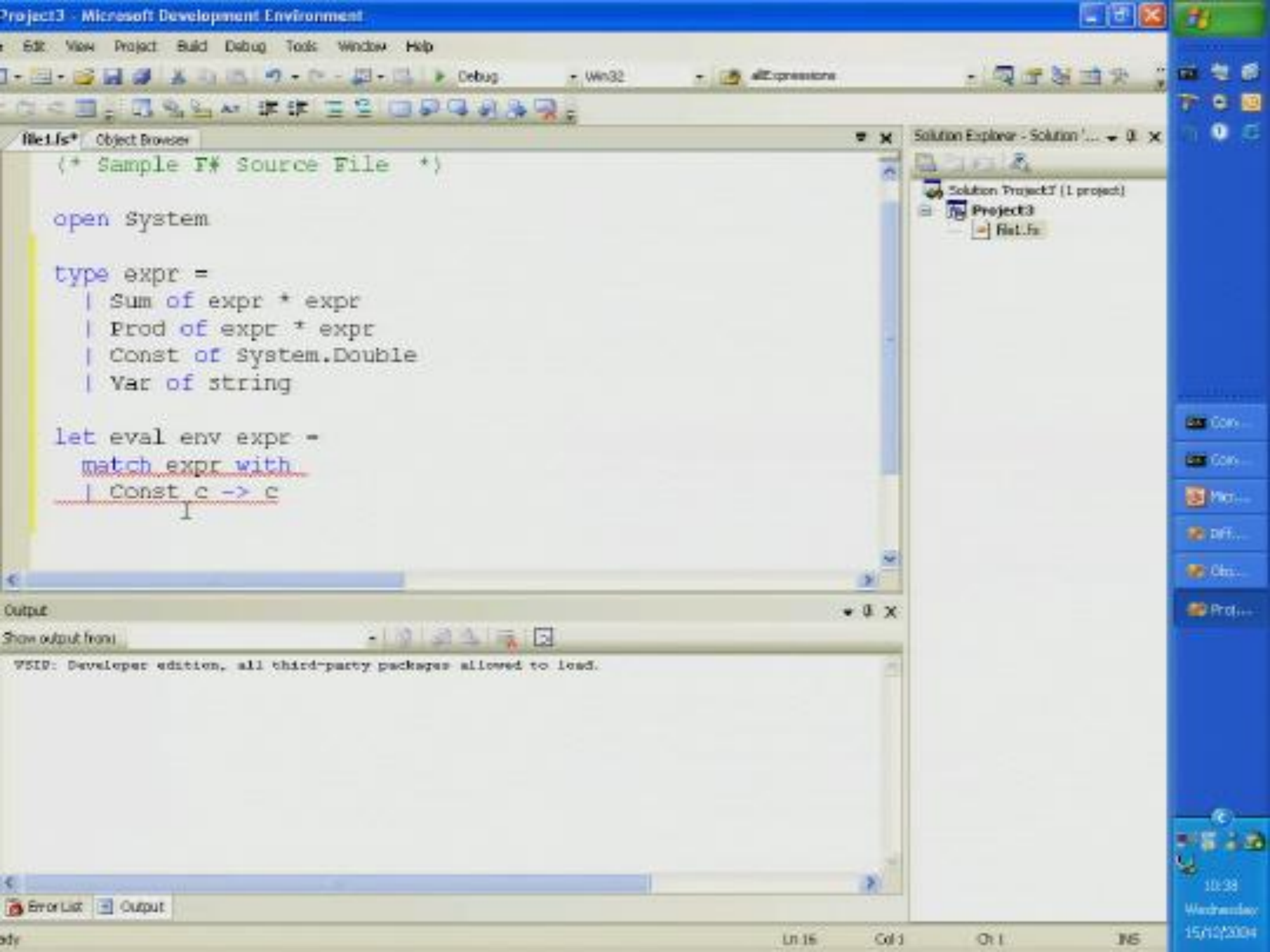
- DEMO

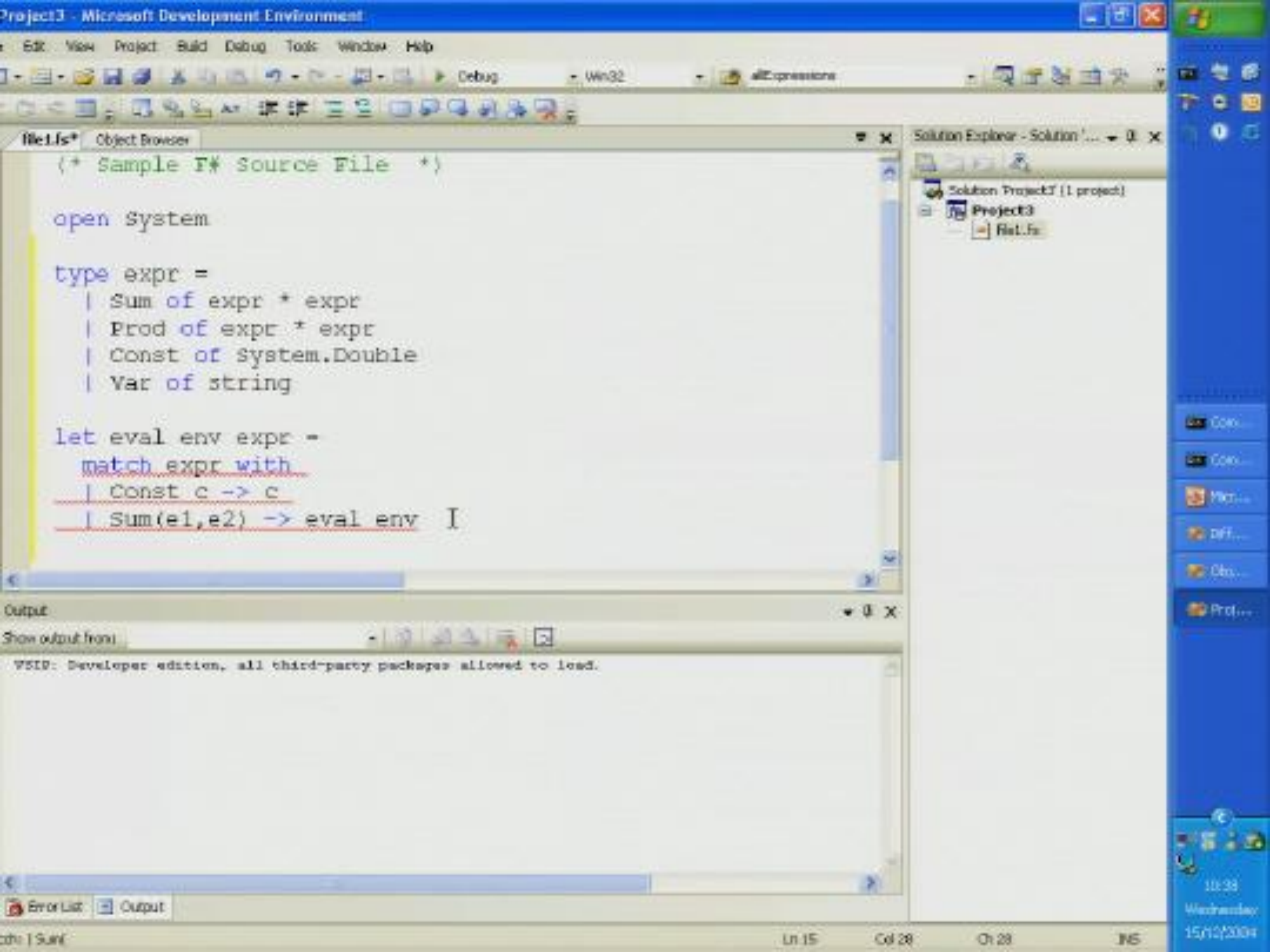
- 1. Write a symbolic differentiator
- 2. Write an evaluation function
- 3. Make it extensible with new operators
- 4. Display sample plots on a form
- 5. Use the differentiator from C#

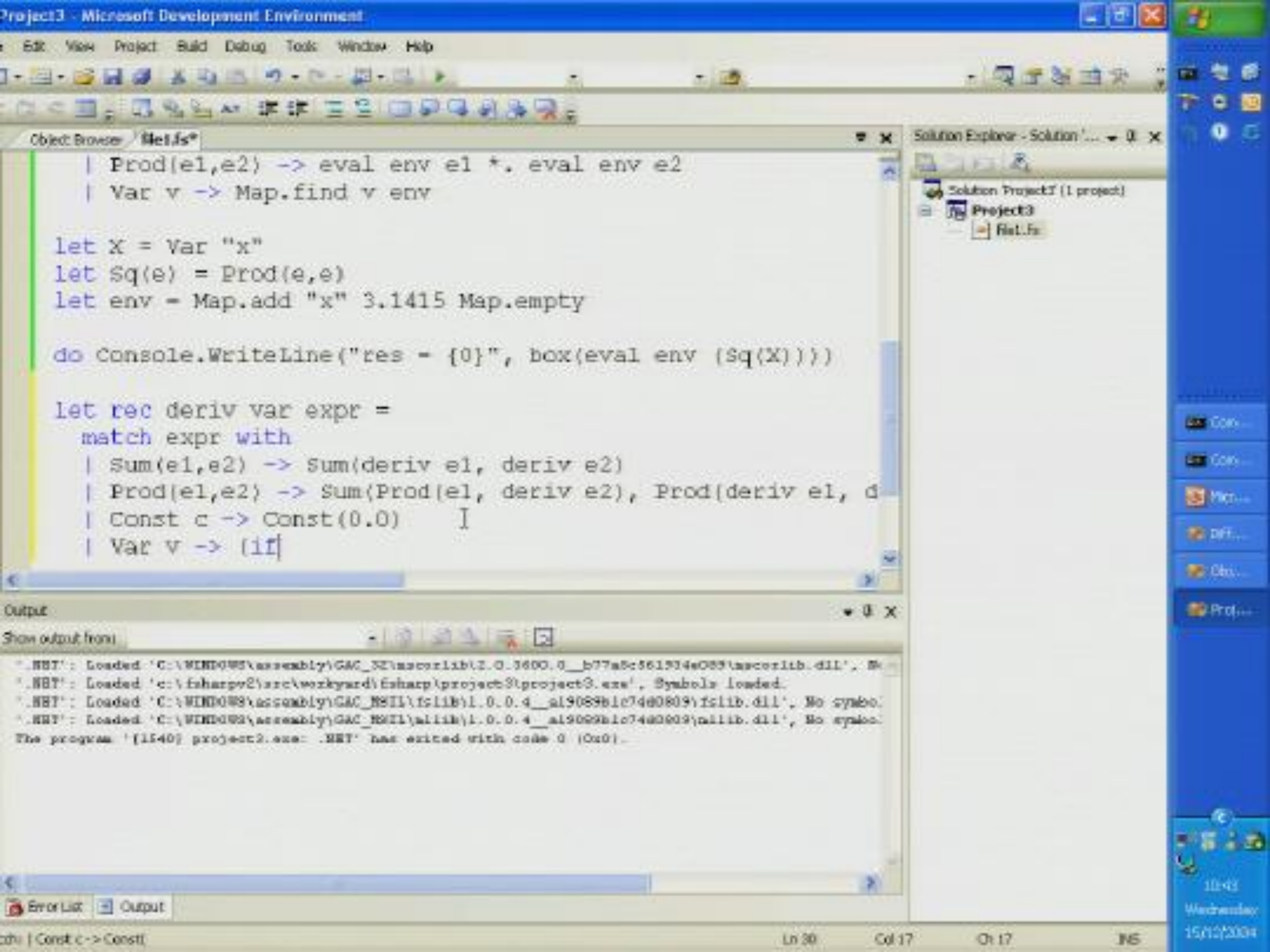

Documents
QuickTime Player
SPACE 2004_files
rescbg
Die Bahn - Bestätigung
dir21dev
nsrc-dsyme
Computer
RealOne Player
videos
RealOnePla...
SPACE 2004
dirx86-3
nsrc-dsyme (large)
cycle Bin
buildreport
canot08_web
T-Online-50...
constrained
dlan2
nsr-ts
Cygwin
Barbara
jacobs97tut...
VSIP05B1.DL
Space 2004 location
dsyme0
Shortcut to FY04
a bug
BarcelonaPo...
serrano
constrained.cs
fsharp-nsr-v1
dsyme0 - large
Shortcut to RSSBandit
fsharp-200...
algplan-why
Indigo
nsr-fsharp-v2
ia64 rolling - large
i-need-your-l...
Tool
Transfer
FY04
TIOZET
RssBanditIns...
newsblurp2004
ia64-rolling
notes
Transfer
ownership types
dtordk_sum2...
22.ps
ReflectionUs...
nsrc-688339
test2
Presentations
graphviz-1.12
paper.ps
MamaMia
nsrc-688339 (large)
Rifx.Jobpack
Presentatio...
ocaml-3.08-win
Rog-O-Matic...
amd64-rolling
nsrc-688619

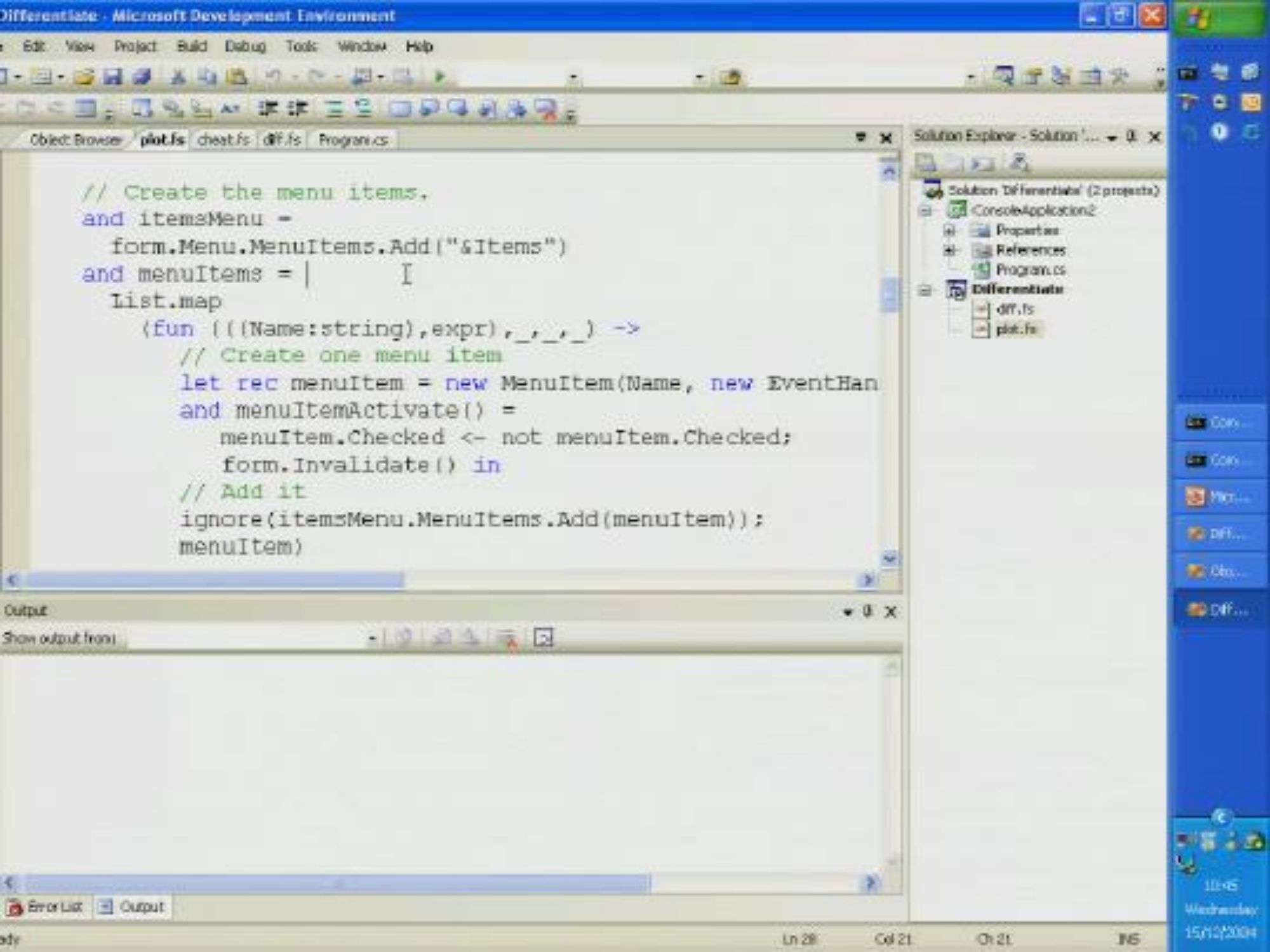

Object Browser - Microsoft Development Environment
Edit View Tools Window Help
Object Browser
Browse: All Components
<Search>
differentiate
fslib
nlilib
mscorlib
System
System.Configu
System.Data
System.Deployn
System.Design
System.Drawing
System.Drawing
System.Enterpri
System.Manage
System.Runtime
Solution Explorer
Output
Show output from:
Error List
Output

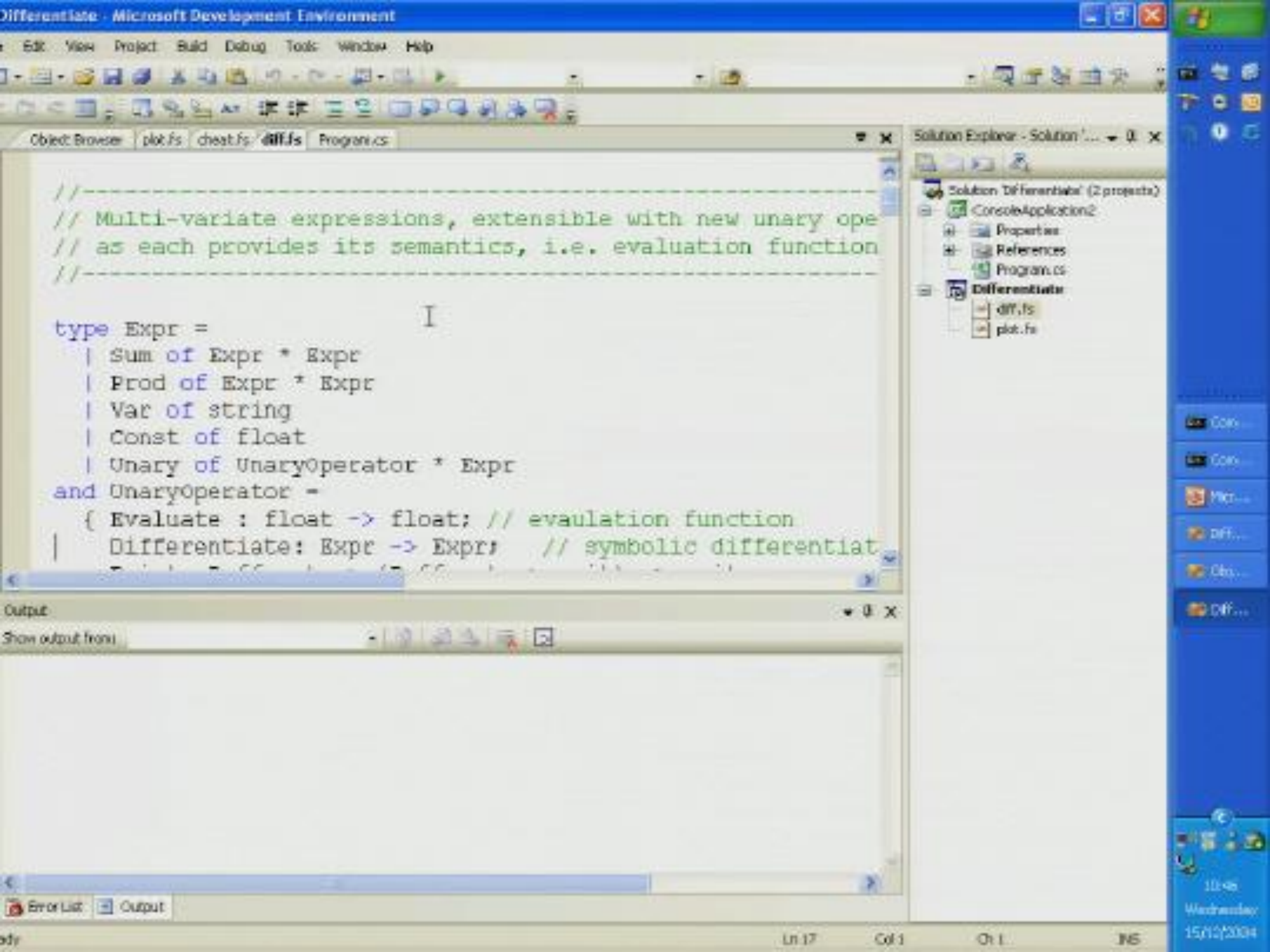

Project3 - Microsoft Development Environment
Edit View Project Build Debug Tools Window Help
Debug
Win32
File1.fs*
Object Browser
(* Sample F# Source File *)
open System
type expr =
| Sum of expr * expr
| Prod of expr * expr
| Const of System.Double
| Var of string
let eval env expr =
match expr with
| Const c
Solution Explorer - Solution '...
Solution 'Project3' (1 project)
Project3
File1.fs
Output
Show output from:
VSIP: Developer edition, all third-party packages allowed to load.
Error List
Output
Ln 14
Col 12
Ch 12
INS

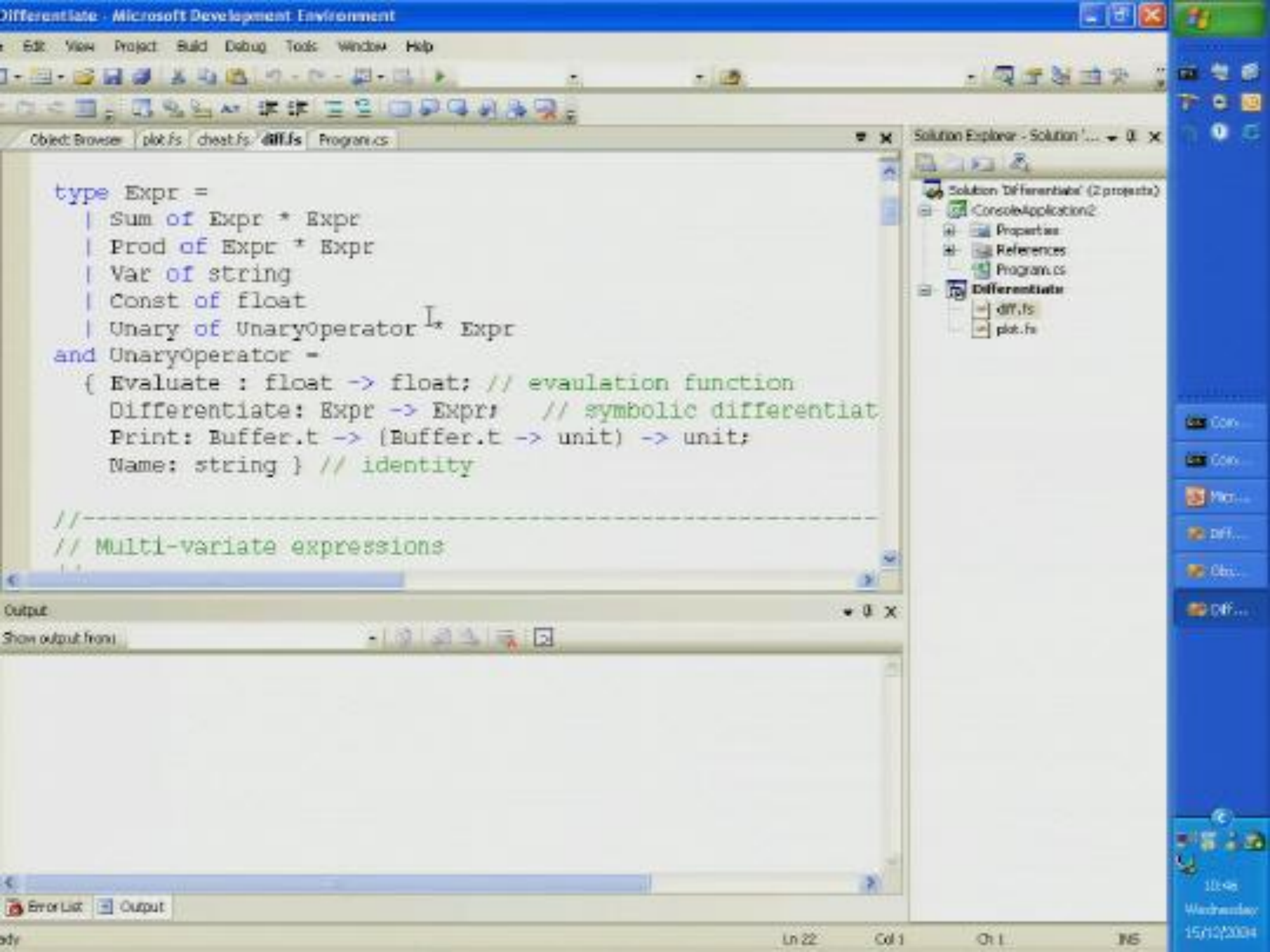

Project3 - Microsoft Development Environment
Edit View Project Build Debug Tools Window Help
Debug
Win32
File1.fs*
Object Browser
(* Sample F# Source File *)
open System
type expr =
| Sum of expr * expr
| Prod of expr * expr
| Const of System.Double
| Var of string
let eval env expr =
match expr with
| Const c -> c
Solution Explorer - Solution '...
Solution 'Project3' (1 project)
Project3
File1.fs
Output
Show output from:
VSIP: Developer edition, all third-party packages allowed to load.
Error List
Output
Ln 16
Col 1
Ch 1
INS

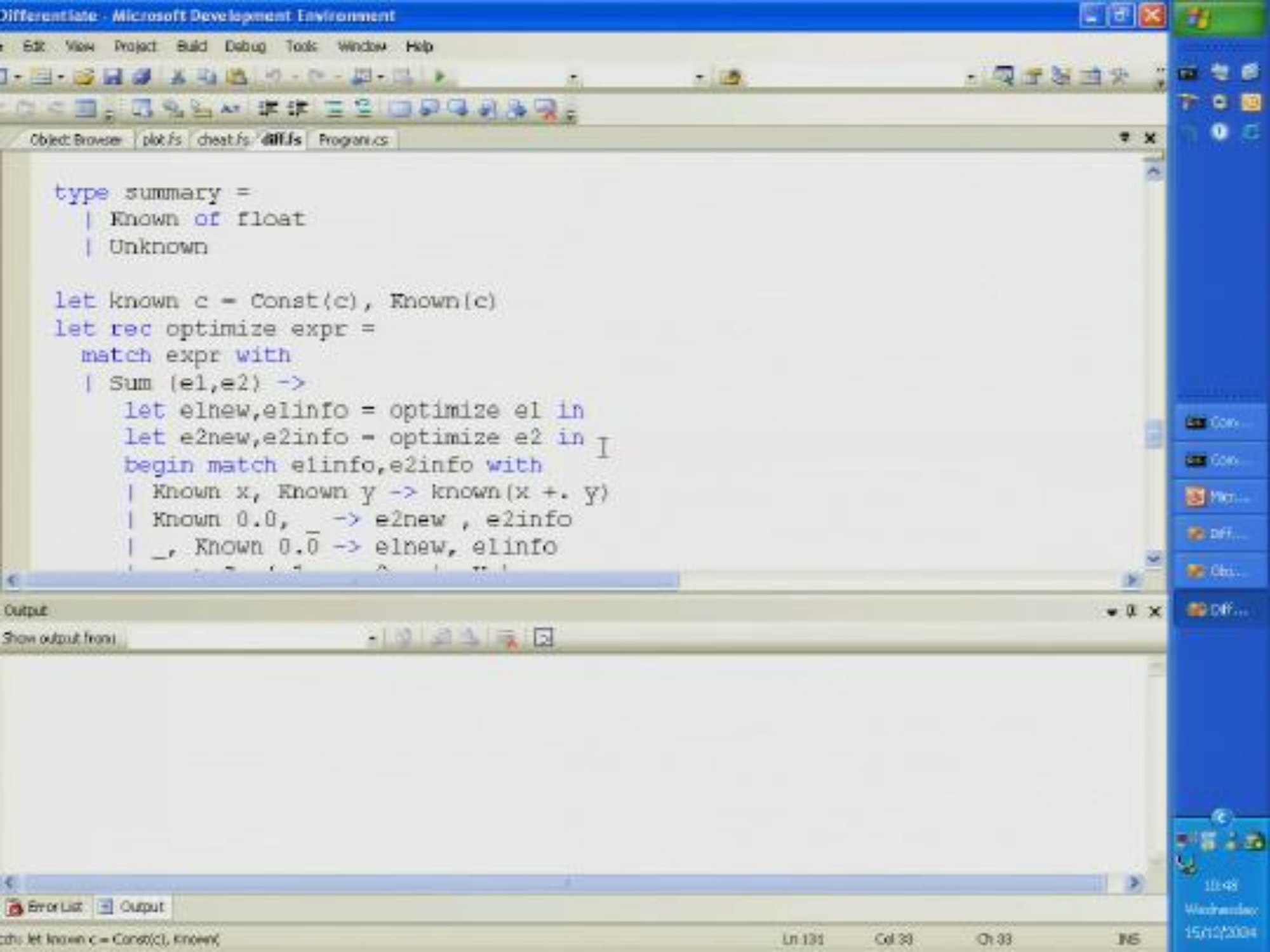

Project3 - Microsoft Development Environment
File Edit View Project Build Debug Tools Window Help
Debug
Win32
File1.fs* Object Browser
(* Sample F# Source File *)
open System
type expr =
| Sum of expr * expr
| Prod of expr * expr
| Const of System.Double
| Var of string
let eval env expr =
match expr with
| Const c -> c
| Sum(e1,e2) -> eval env
Solution Explorer - Solution '...
Solution 'Project3' (1 project)
Project3
File1.fs
Output
Show output from:
VSIP: Developer edition, all third-party packages allowed to load.
Error List
Output
Ln 15
Col 28
Ch 28
INS

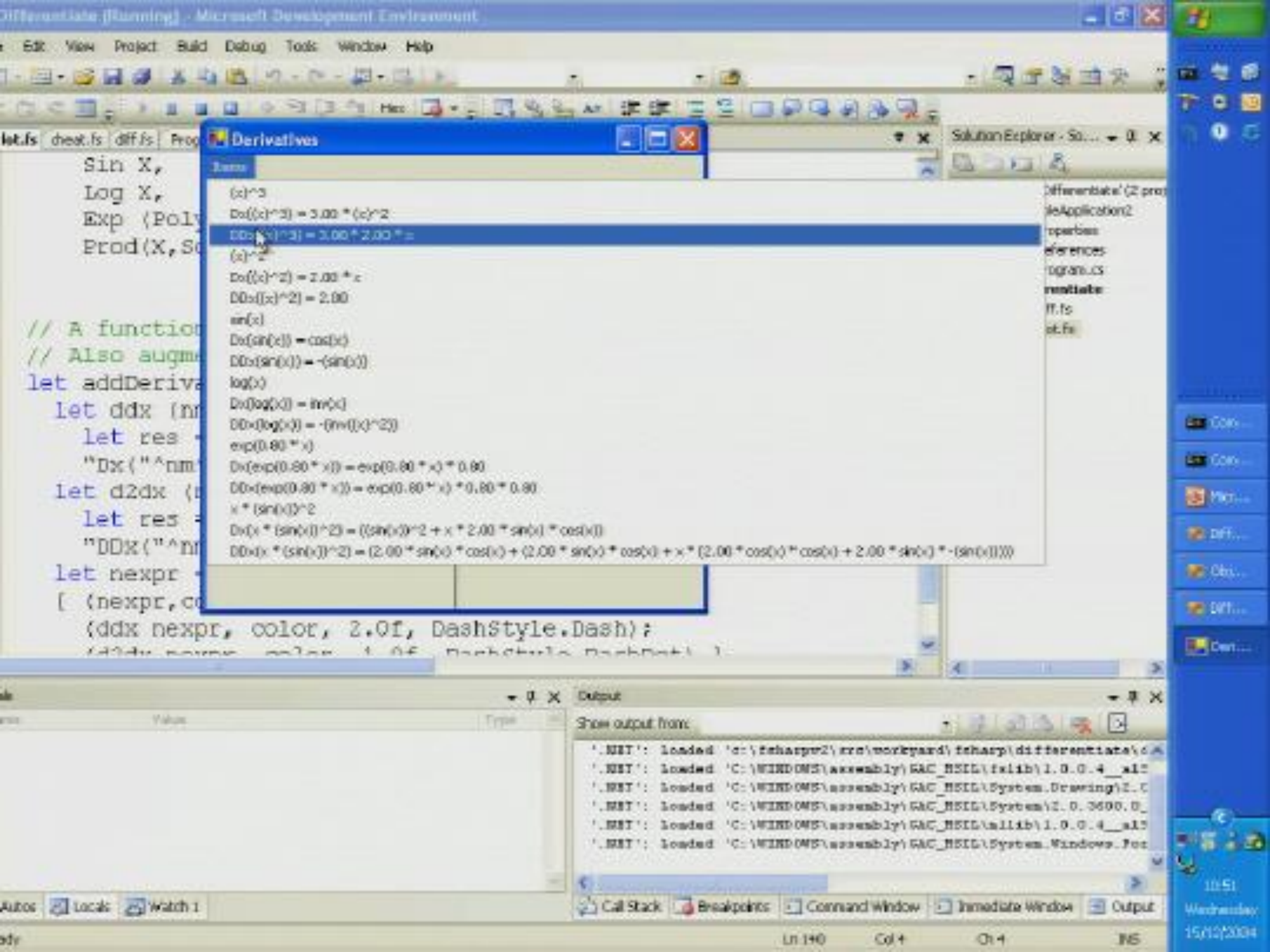

Project3 - Microsoft Development Environment
Edit View Project Build Debug Tools Window Help
Object Browser
Rat1.fs*
| Prod(e1,e2) -> eval env e1 *. eval env e2
| Var v -> Map.find v env
let X = Var "x"
let Sq(e) = Prod(e,e)
let env = Map.add "x" 3.1415 Map.empty
do Console.WriteLine("res = {0}", box(eval env (Sq(X))))
let rec deriv var expr =
match expr with
| Sum(e1,e2) -> Sum(deriv e1, deriv e2)
| Prod(e1,e2) -> Sum(Prod(e1, deriv e2), Prod(deriv e1, d
| Const c -> Const(0.0)
| Var v -> (if
Solution Explorer - Solution '...
Solution 'Project3' (1 project)
Project3
Rat1.fs
Output
Show output from:
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.3600.0__b77a5c561934e089\mscorlib.dll', N
'.NET': Loaded 'c:\fsharpv2\src\workyard\fsharp\project3\project3.exe', Symbols loaded.
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_MSIL\fslib\1.0.0.4__a19089b1c74d0809\fslib.dll', No symbo
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_MSIL\mllib\1.0.0.4__a19089b1c74d0809\mllib.dll', No symbo
The program '[1540] project2.exe: .NET' has exited with code 0 (0x0).
Error List
Output
Ln 30
Col 17
Ch 17
INS

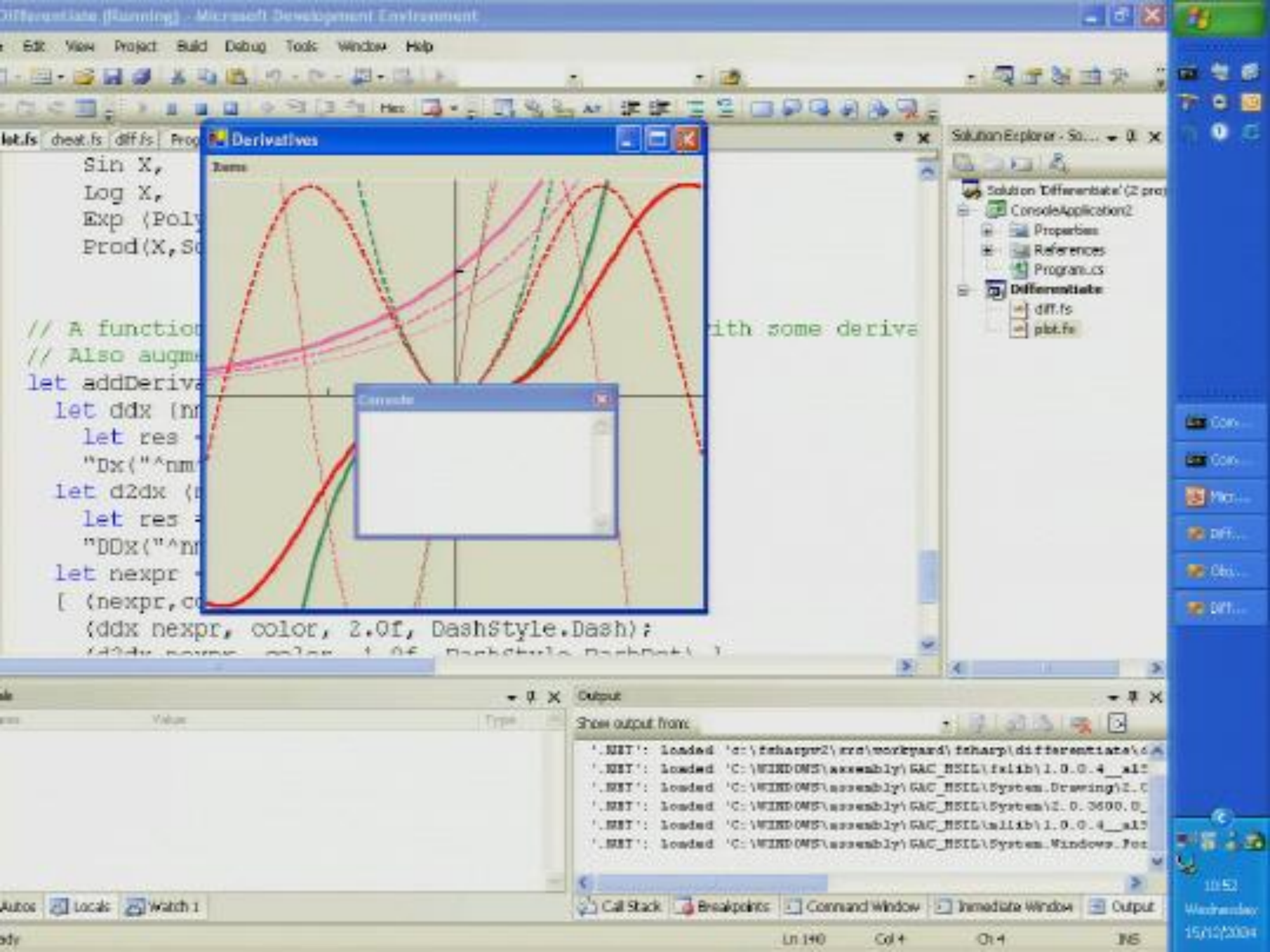

Differentiate - Microsoft Development Environment
File Edit View Project Build Debug Tools Window Help
Object Browser
plot.fs
chest.fs
diff.fs
Program.cs
// Create the menu items.
and itemsMenu =
form.Menu.MenuItems.Add("&Items")
and menuItems =
List.map
(fun (((Name:string),expr),_,_,_) ->
// Create one menu item
let rec menuItem = new MenuItem(Name, new EventHan
and menuItemActivate() =
menuItem.Checked <- not menuItem.Checked;
form.Invalidate() in
// Add it
ignore(itemsMenu.MenuItems.Add(menuItem));
menuItem)
Solution Explorer - Solution '...
Solution 'Differentiate' (2 projects)
ConsoleApplication2
Properties
References
Program.cs
Differentiate
diff.fs
plot.fs
Output
Show output from:
Error List
Output
Ln 28
Col 21
Ch 21
INS

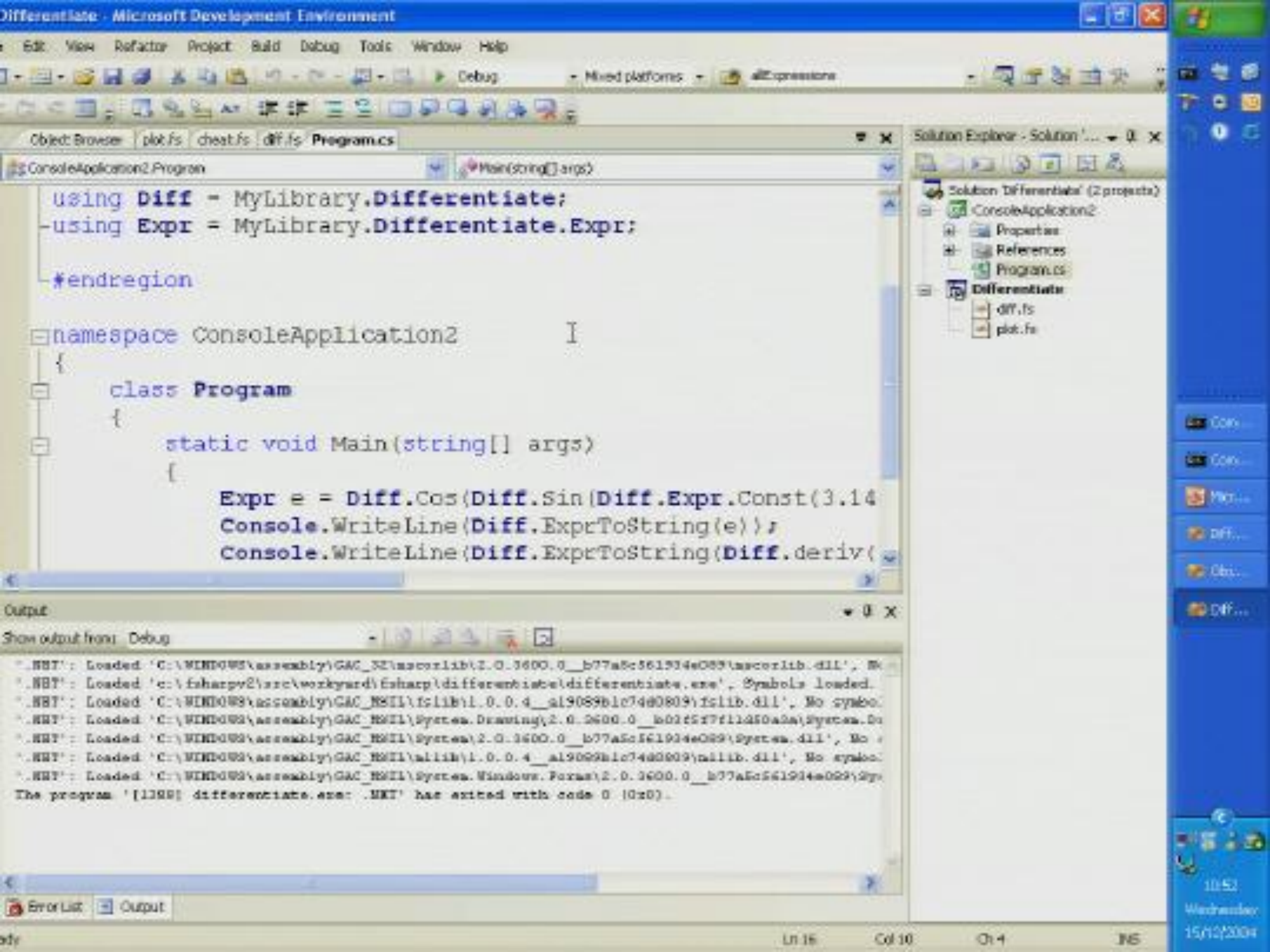

Differentiate - Microsoft Development Environment
File Edit View Project Build Debug Tools Window Help
Object Browser
plot.fs
cheat.fs
diff.fs
Program.cs
//--
// Multi-variate expressions, extensible with new unary ope
// as each provides its semantics, i.e. evaluation function
//--
type Expr =
| Sum of Expr * Expr
| Prod of Expr * Expr
| Var of string
| Const of float
| Unary of UnaryOperator * Expr
and UnaryOperator =
{ Evaluate : float -> float; // evaulation function
Differentiate: Expr -> Expr; // symbolic differentiat
Solution Explorer - Solution '...
Solution 'Differentiate' (2 projects)
ConsoleApplication2
Properties
References
Program.cs
Differentiate
diff.fs
plot.fs
Output
Show output from:
Error List
Output
Ln 17
Col 1
Ch 1
INS

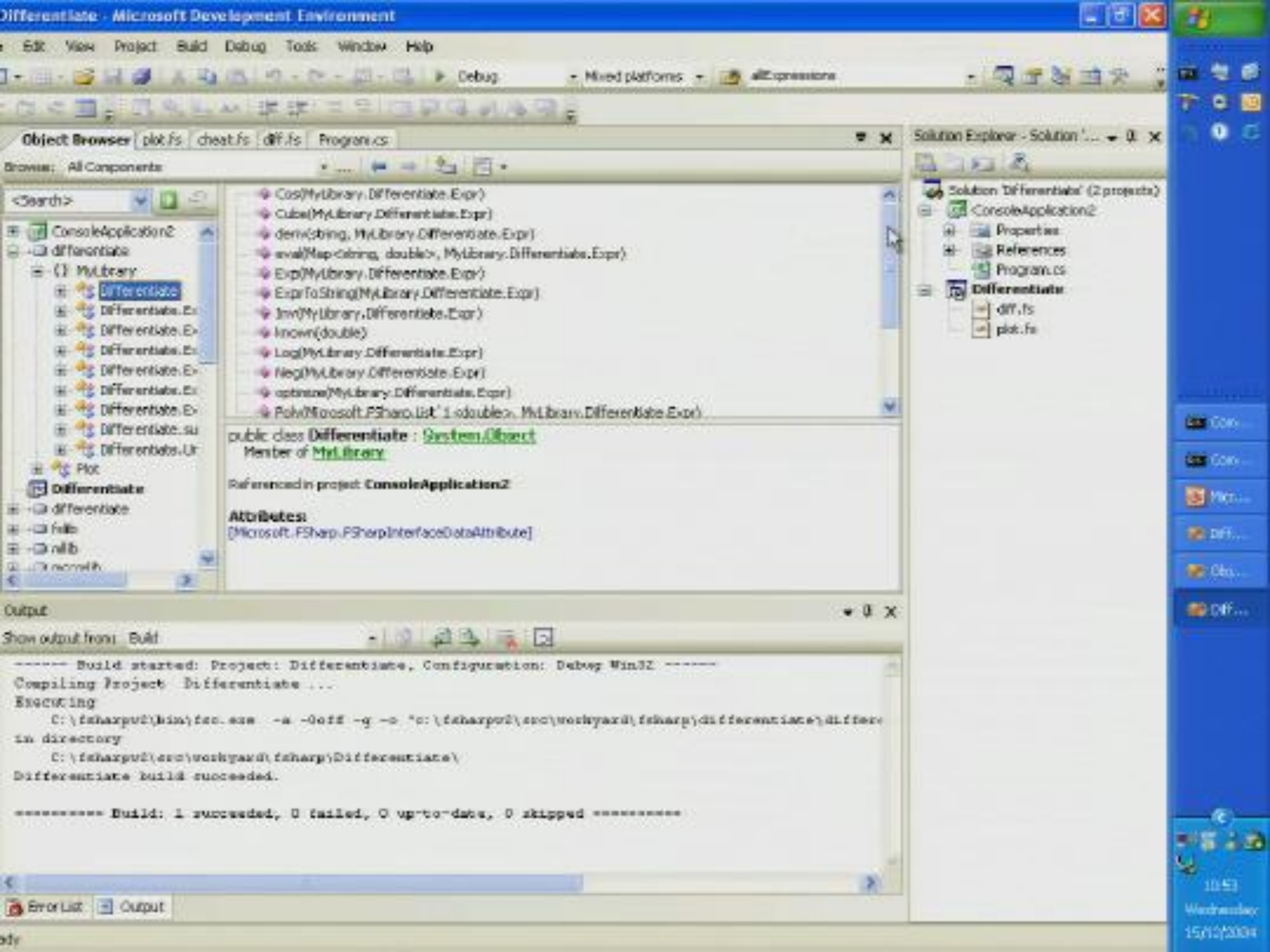

Differentiate - Microsoft Development Environment
File Edit View Project Build Debug Tools Window Help
Object Browser
plot.fs
chest.fs
diff.fs
Program.cs
type Expr =
| Sum of Expr * Expr
| Prod of Expr * Expr
| Var of string
| Const of float
| Unary of UnaryOperator * Expr
and UnaryOperator =
{ Evaluate : float -> float; // evaulation function
Differentiate: Expr -> Expr; // symbolic differentiat
Print: Buffer.t -> (Buffer.t -> unit) -> unit;
Name: string } // identity
//
// Multi-variate expressions
Solution Explorer - Solution '...
Solution 'Differentiate' (2 projects)
ConsoleApplication2
Properties
References
Program.cs
Differentiate
diff.fs
plot.fs
Output
Show output from:
Error List
Output
Ln 22
Col 1
Ch 1
INS

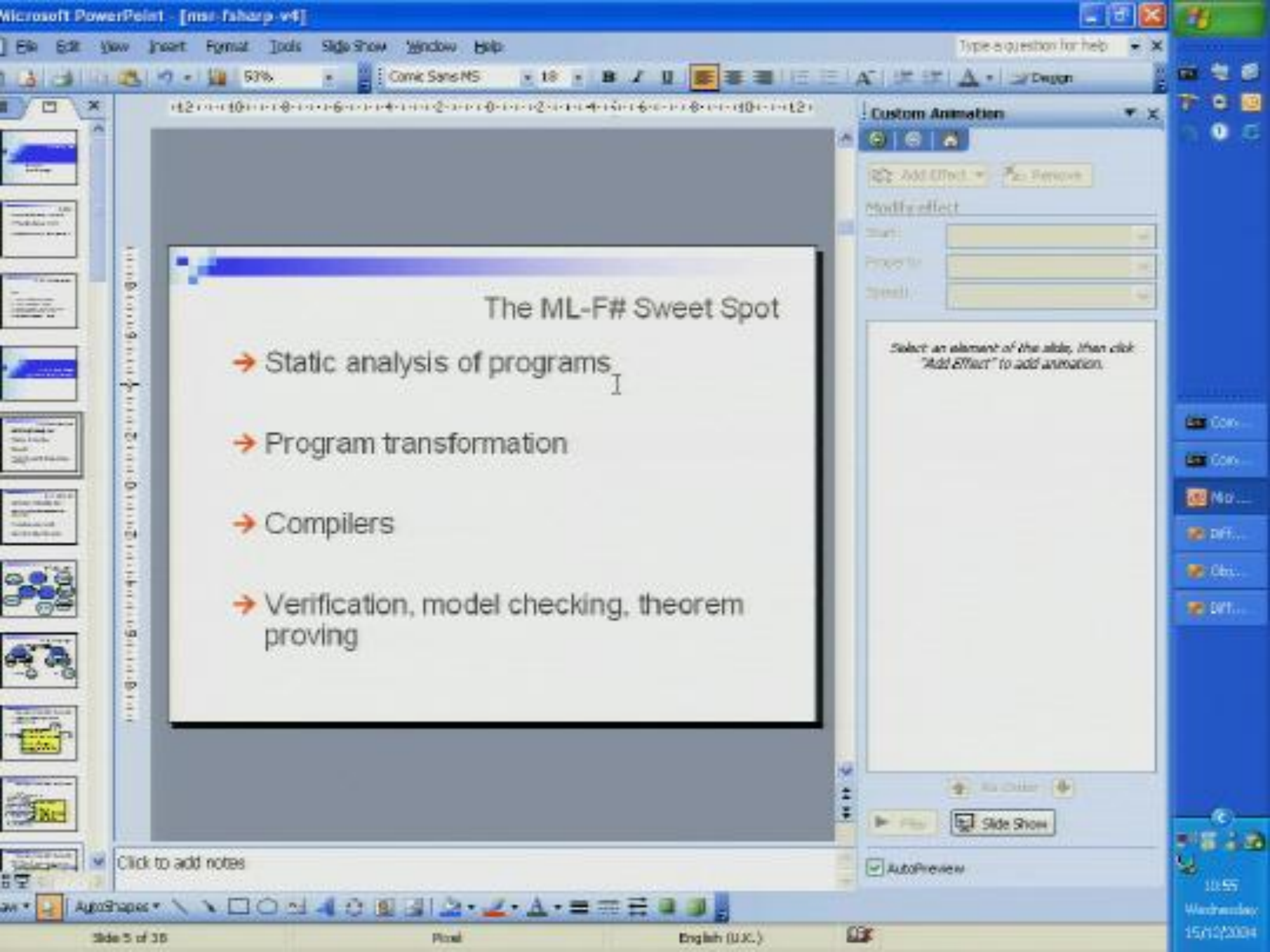

Differentiate - Microsoft Development Environment
Edit View Project Build Debug Tools Window Help
Object Browser plot.fs cheat.fs diff.fs Program.cs
type summary =
| Known of float
| Unknown
let known c = Const(c), Known(c)
let rec optimize expr =
match expr with
| Sum (e1,e2) ->
let e1new,e1info = optimize e1 in
let e2new,e2info = optimize e2 in
begin match e1info,e2info with
| Known x, Known y -> known(x +. y)
| Known 0.0, _ -> e2new , e2info
| _, Known 0.0 -> e1new, e1info
Output
Show output from:
Error List
Output
Ln 131
Col 33
Ch 33
INS


Differentiate (Running) - Microsoft Development Environment
Edit View Project Build Debug Tools Window Help
Derivatives
Items
(x)^3
D×((x)^3) = 3.00 * (x)^2
DD×((x)^3) = 3.00 * 2.00 * x
(x)^2
D×((x)^2) = 2.00 * x
DD×((x)^2) = 2.00
sin(x)
D×(sin(x)) = cos(x)
DD×(sin(x)) = -(sin(x))
log(x)
D×(log(x)) = inv(x)
DD×(log(x)) = -(inv(x)^2))
exp(0.80 * x)
D×(exp(0.80 * x)) = exp(0.80 * x) * 0.80
DD×(exp(0.80 * x)) = exp(0.80 * x) * 0.80 * 0.80
x * (sin(x))^2
D×(x * (sin(x))^2) = ((sin(x))^2 + x * 2.00 * sin(x) * cos(x))
DD×(x * (sin(x))^2) = (2.00 * sin(x) * cos(x) + (2.00 * sin(x) * cos(x) + x * (2.00 * cos(x) * cos(x) + 2.00 * sin(x) * -(sin(x)))))
Solution Explorer - So...
Output
Show output from:
Call Stack
Breakpoints
Command Window
Immediate Window
Output
Autos
Locals
Watch 1
Ln 140
Col 4
Ch 4
INS


Derivatives
Sin X,
Log X,
Exp (Poly
Prod(X,Sc
// A functio
// Also augm
let addDeriva
let ddx (nm
let res
"Dx("^nm
let d2dx (n
let res
"DDx("^nm
let nexpr
[(nexpr,co
(ddx nexpr, color, 2.0f, DashStyle.Dash);
Solution Explorer - So...
Solution 'Differentiate' (2 proj
ConsoleApplication2
Properties
References
Program.cs
Differentiate
diff.fs
plot.fs
Output
Show output from:
Call Stack
Breakpoints
Command Window
Immediate Window
Output
Ln 140
Col 4
Ch 4


Differentiate - Microsoft Development Environment
Edit View Refactor Project Build Debug Tools Window Help
Debug
Mixed Platforms
Object Browser
plot.fs
cheat.fs
diff.fs
Program.cs
ConsoleApplication2.Program
Main(string[] args)
using Diff = MyLibrary.Differentiate;
using Expr = MyLibrary.Differentiate.Expr;
#endregion
namespace ConsoleApplication2
{
class Program
{
static void Main(string[] args)
{
Expr e = Diff.Cos(Diff.Sin(Diff.Expr.Const(3.14
Console.WriteLine(Diff.ExprToString(e));
Console.WriteLine(Diff.ExprToString(Diff.deriv(
Solution Explorer - Solution '...
Solution 'Differentiate' (2 projects)
ConsoleApplication2
Properties
References
Program.cs
Differentiate
diff.fs
plot.fs
Output
Show output from: Debug
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_32\mscorlib\2.0.3600.0__b77a5c561934e089\mscorlib.dll', No
'.NET': Loaded 'c:\fsharpv2\src\workyard\fsharp\differentiate\differentiate.exe', Symbols loaded.
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_MSIL\fslib\1.0.0.4__a19089b1c74d0809\fslib.dll', No symbo
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_MSIL\System.Drawing\2.0.3600.0__b03f5f7f11d50a3a\System.Dr
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_MSIL\System\2.0.3600.0__b77a5c561934e089\System.dll', No
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_MSIL\mllib\1.0.0.4__a19089b1c74d0809\mllib.dll', No symbo
'.NET': Loaded 'C:\WINDOWS\assembly\GAC_MSIL\System.Windows.Forms\2.0.3600.0__b77a5c561934e089\Sy
The program '[1388] differentiate.exe: .NET' has exited with code 0 (0x0).
Error List
Output
Ln 16
Col 10
Ch 4
INS
10:52
Wednesday
15/12/2004

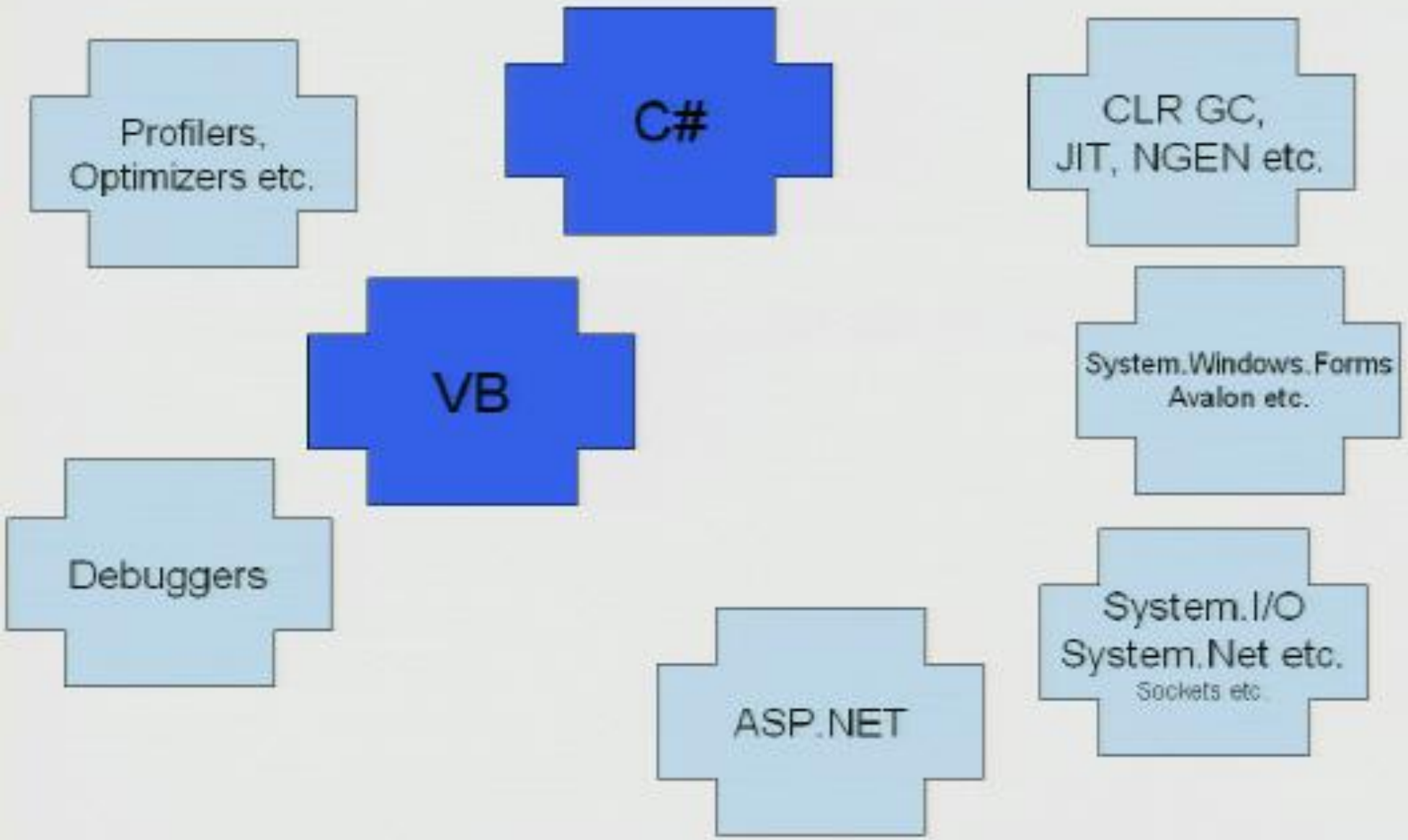

Differentiate - Microsoft Development Environment
Edit View Project Build Debug Tools Window Help
Debug
Mixed Platforms
Object Browser
plot.fs
chest.fs
diff.fs
Program.cs
Browse: All Components
<Search>
ConsoleApplication2
differentiate
MyLibrary
Differentiate
Plot
Differentiate
Cos(MyLibrary.Differentiate.Expr)
Cube(MyLibrary.Differentiate.Expr)
deriv(string, MyLibrary.Differentiate.Expr)
eval(Map<string, double>, MyLibrary.Differentiate.Expr)
Exp(MyLibrary.Differentiate.Expr)
ExprToString(MyLibrary.Differentiate.Expr)
Inv(MyLibrary.Differentiate.Expr)
known(double)
Log(MyLibrary.Differentiate.Expr)
Neg(MyLibrary.Differentiate.Expr)
optimize(MyLibrary.Differentiate.Expr)
Poly(Microsoft.FSharp.List`1<double>, MyLibrary.Differentiate.Expr)
public class Differentiate : System.Object
Member of MyLibrary
Referenced in project ConsoleApplication2
Attributes:
[Microsoft.FSharp.FSharpInterfaceDataAttribute]
Solution Explorer - Solution '...
Solution 'Differentiate' (2 projects)
ConsoleApplication2
Properties
References
Program.cs
Differentiate
diff.fs
plot.fs
Output
Show output from: Build
------ Build started: Project: Differentiate, Configuration: Debug Win32 ------
Compiling Project Differentiate ...
Executing
C:\fsharp02\bin\fsc.exe -a -Ooff -g -o "c:\fsharp02\src\workyard\fsharp\differentiate\differ
in directory
C:\fsharp02\src\workyard\fsharp\Differentiate\
Differentiate build succeeded.
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
Error List
Output


Microsoft PowerPoint - [msr-fsharp-v4]
The ML-F# Sweet Spot
Static analysis of programs
Program transformation
Compilers
Verification, model checking, theorem proving
Custom Animation
Click to add notes

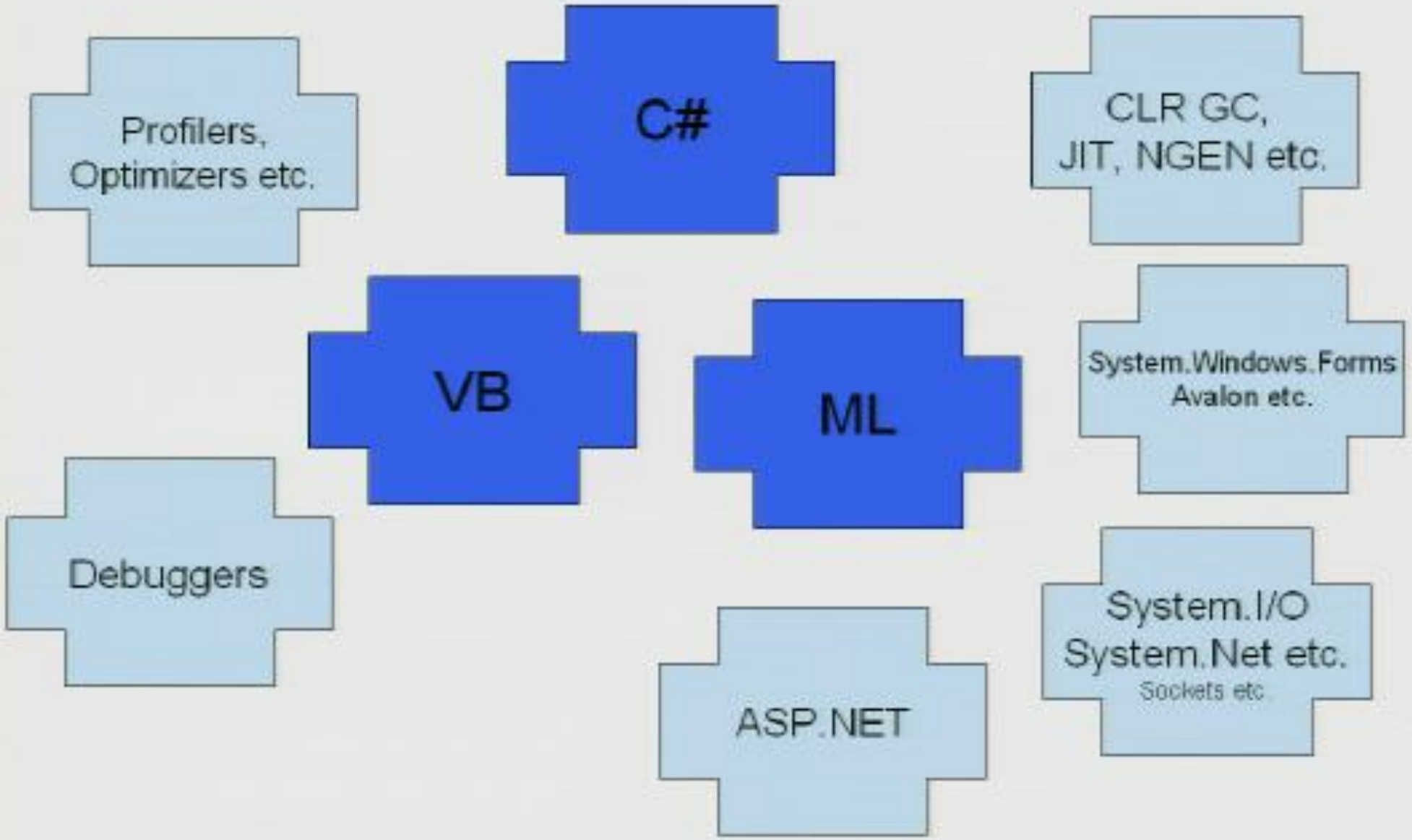
ML in Microsoft

- Static Driver Verifier (Core OS)
- High-level model of FRS (Core File Services)
- Zap theorem prover (MSR)
- KIS race condition finder (MSR)

F# within .NET


Profilers,
Optimizers etc.
C#
CLR GC,
JIT, NGEN etc.
VB
System.Windows.Forms
Avalon etc.
Debuggers
System.I/O
System.Net etc.
Sockets etc.
ASP.NET

F# within .NET


Profilers,
Optimizers etc.
C#
CLR GC,
JIT, NGEN etc.
VB
ML
System.Windows.Forms
Avalon etc.
Debuggers
System.I/O
System.Net etc.
Sockets etc.
ASP.NET

F# as a Language

OCaml

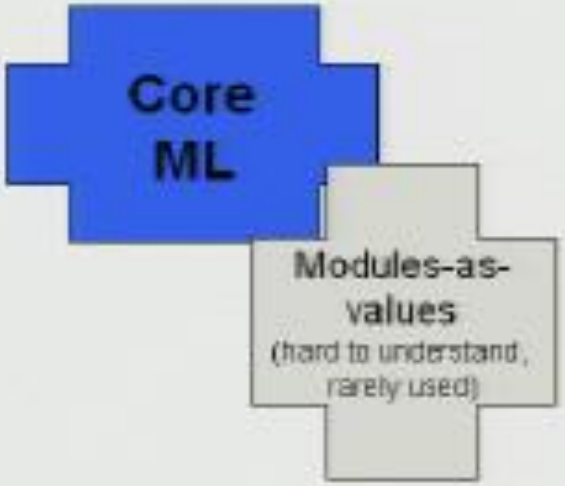
F#

F# as a Language

OCaml

F#

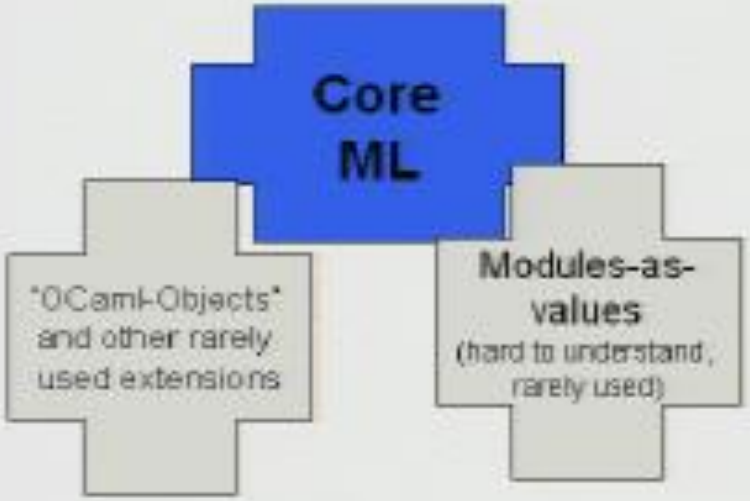
F# as a Language



OCaml

F#

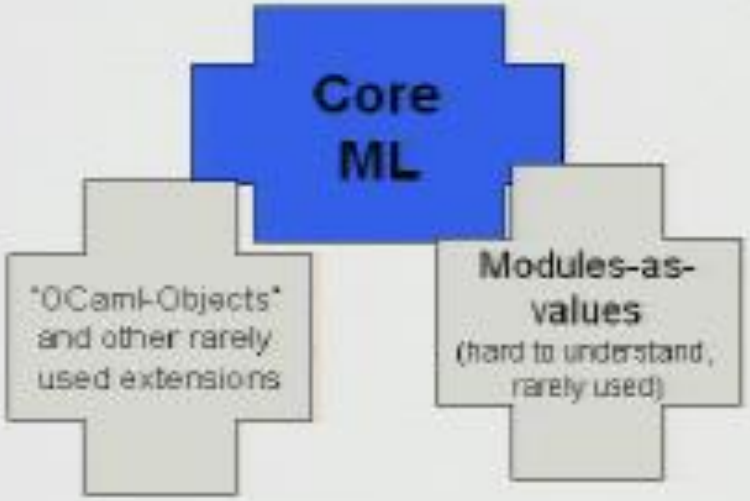
F# as a Language

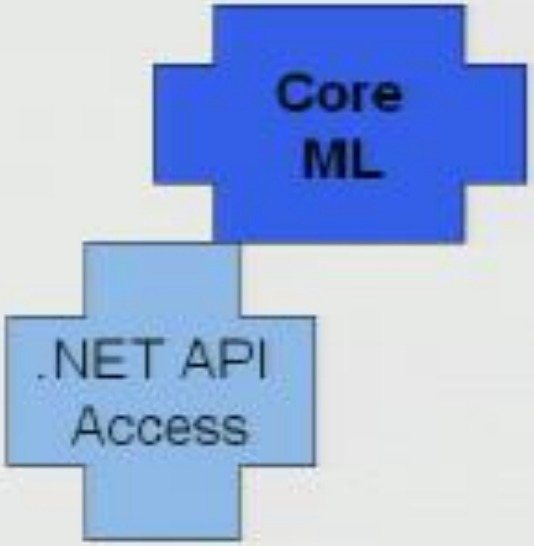


OCaml

F#

F# as a Language

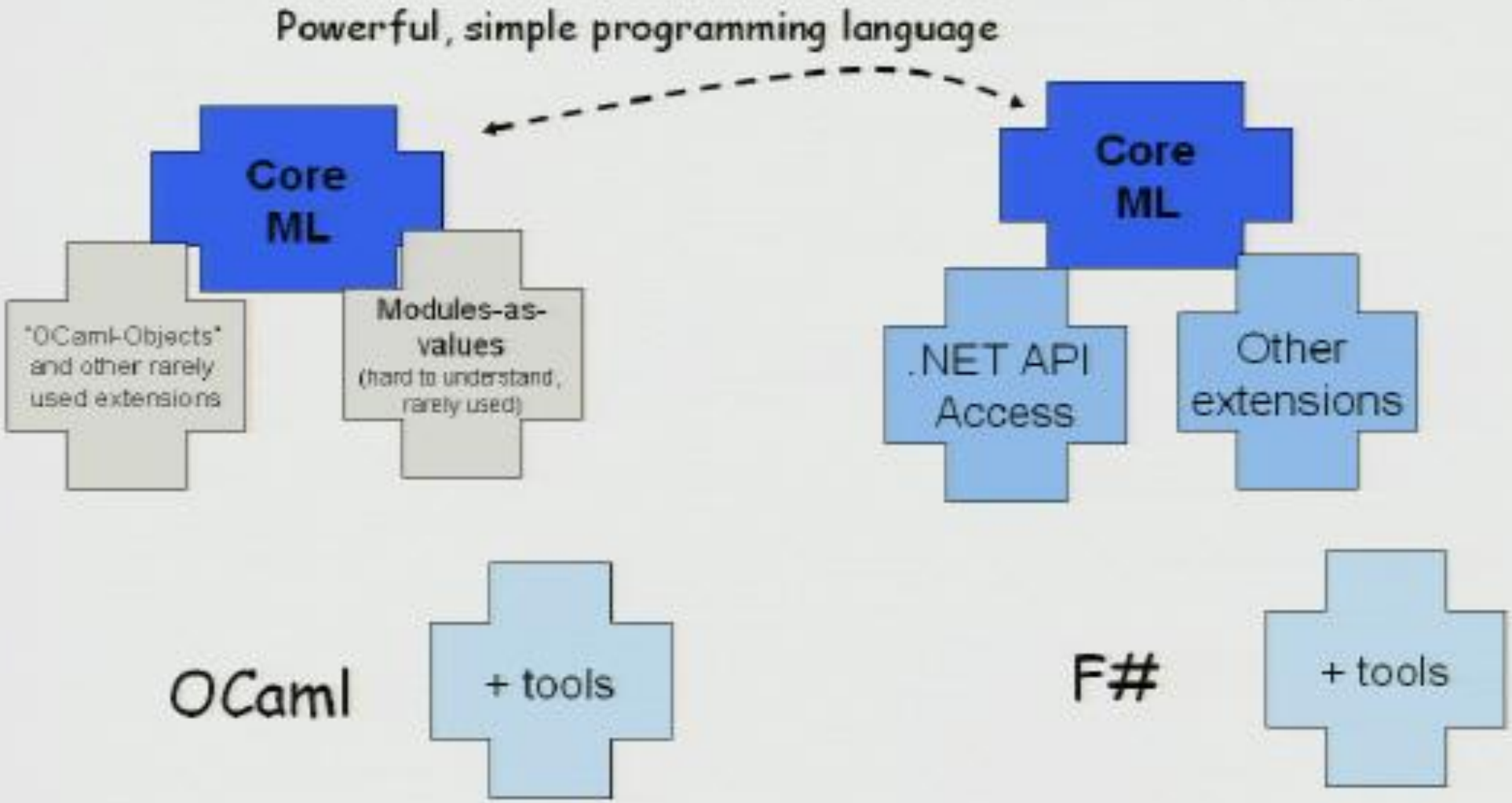




OCaml + tools

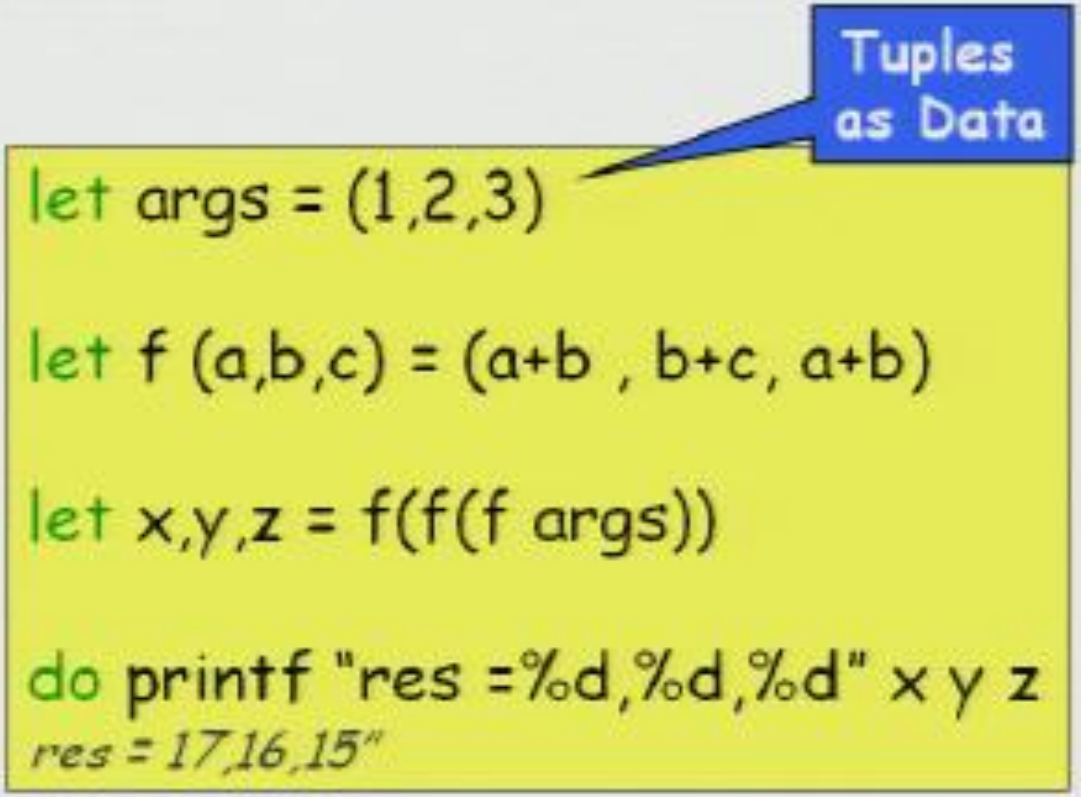
F#

F# as a Language



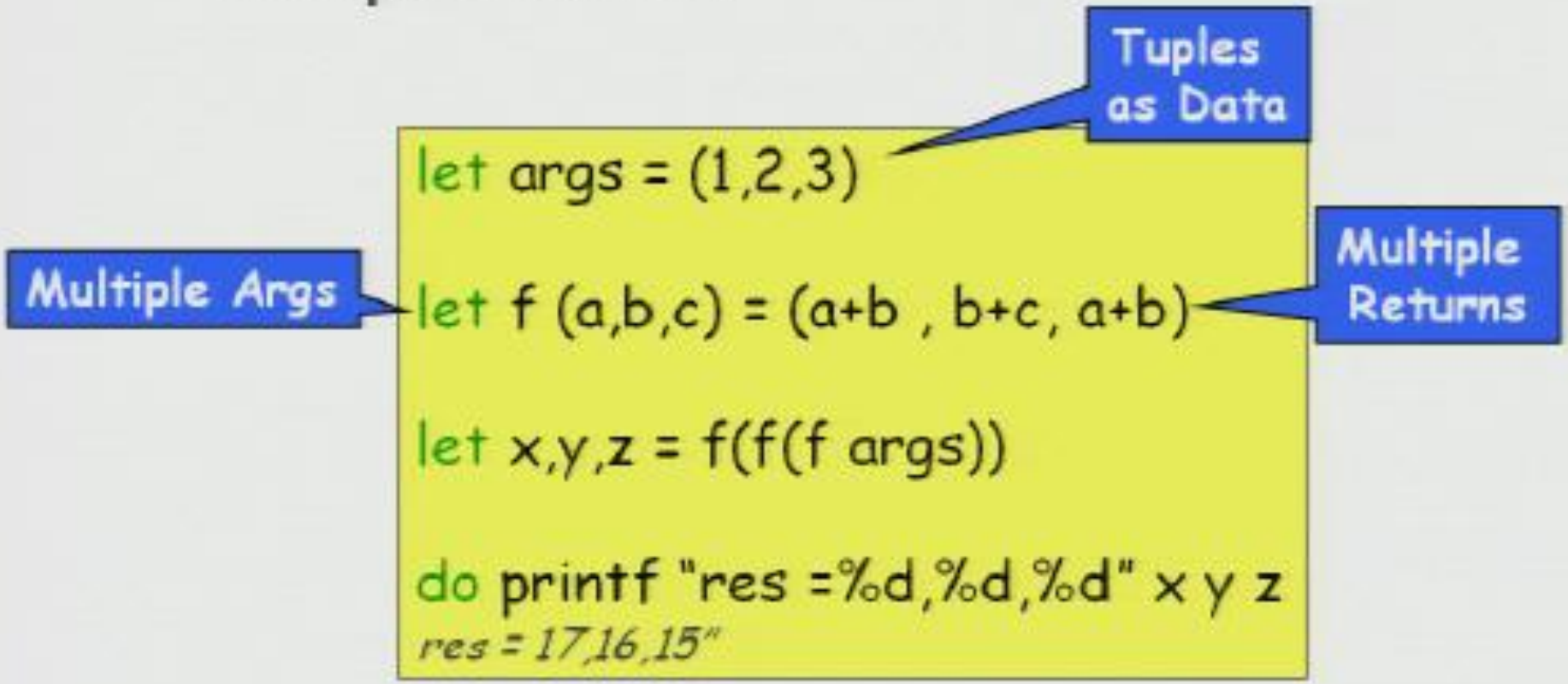
Orthogonal & Unified Constructs

- Tuples = multiple arguments = multiple returns



Orthogonal & Unified Constructs

- Tuples = multiple arguments = multiple returns



Orthogonal & Unified Constructs

- Let "let" simplify your life…

```
let data = (1,2,3)

let f(a,b,c) =
    let sum = a + b + c in
    let g(x) = sum + x*x in
    g(a), g(b), g(c)
```

Orthogonal & Unified Constructs

→ Let "let" simplify your life…

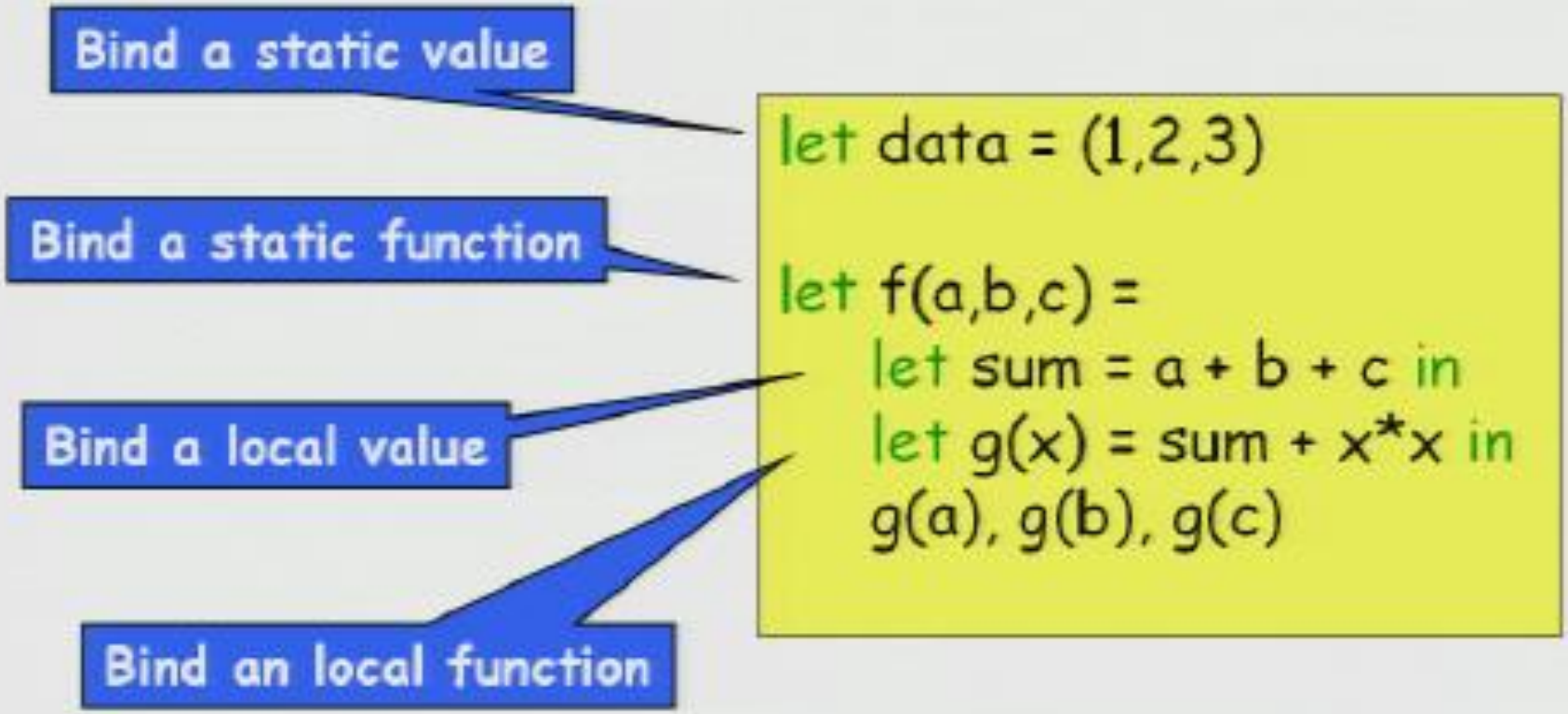
Bind a static value

```
let data = (1,2,3)

let f(a,b,c) =
    let sum = a + b + c in
    let g(x) = sum + x*x in
    g(a), g(b), g(c)
```

Orthogonal & Unified Constructs

→ Let "let" simplify your life…



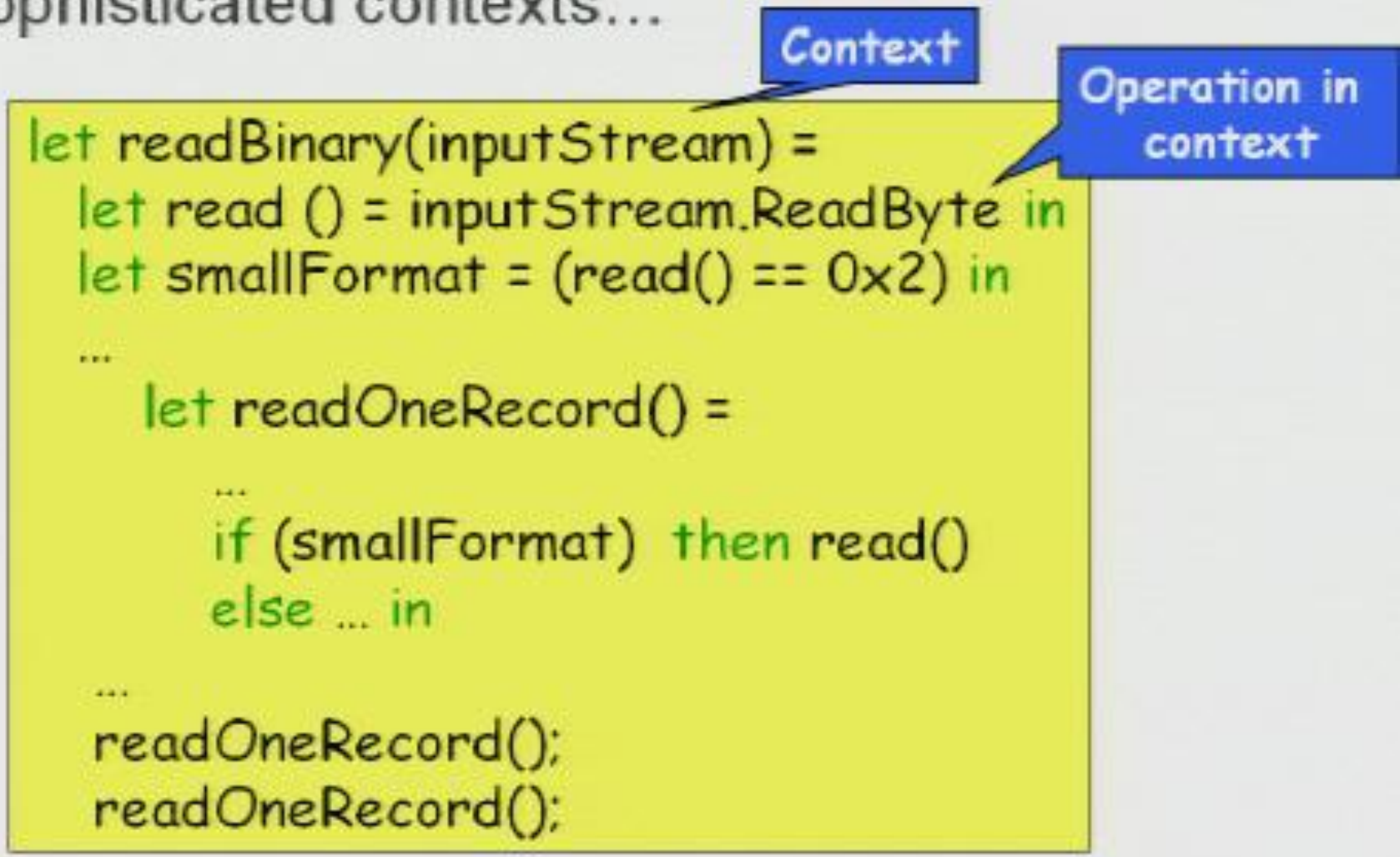
Orthogonal & Unified Constructs

- "let" + "capture": sophisticated operations in sophisticated contexts…

```
let readBinary(inputStream) =
   let read () = inputStream.ReadByte in
   let smallFormat = (read() == 0x2) in
   ...
      let readOneRecord() =
         ...
         if (smallFormat)  then read()
         else ... in
   ...
   readOneRecord();
   readOneRecord();
```

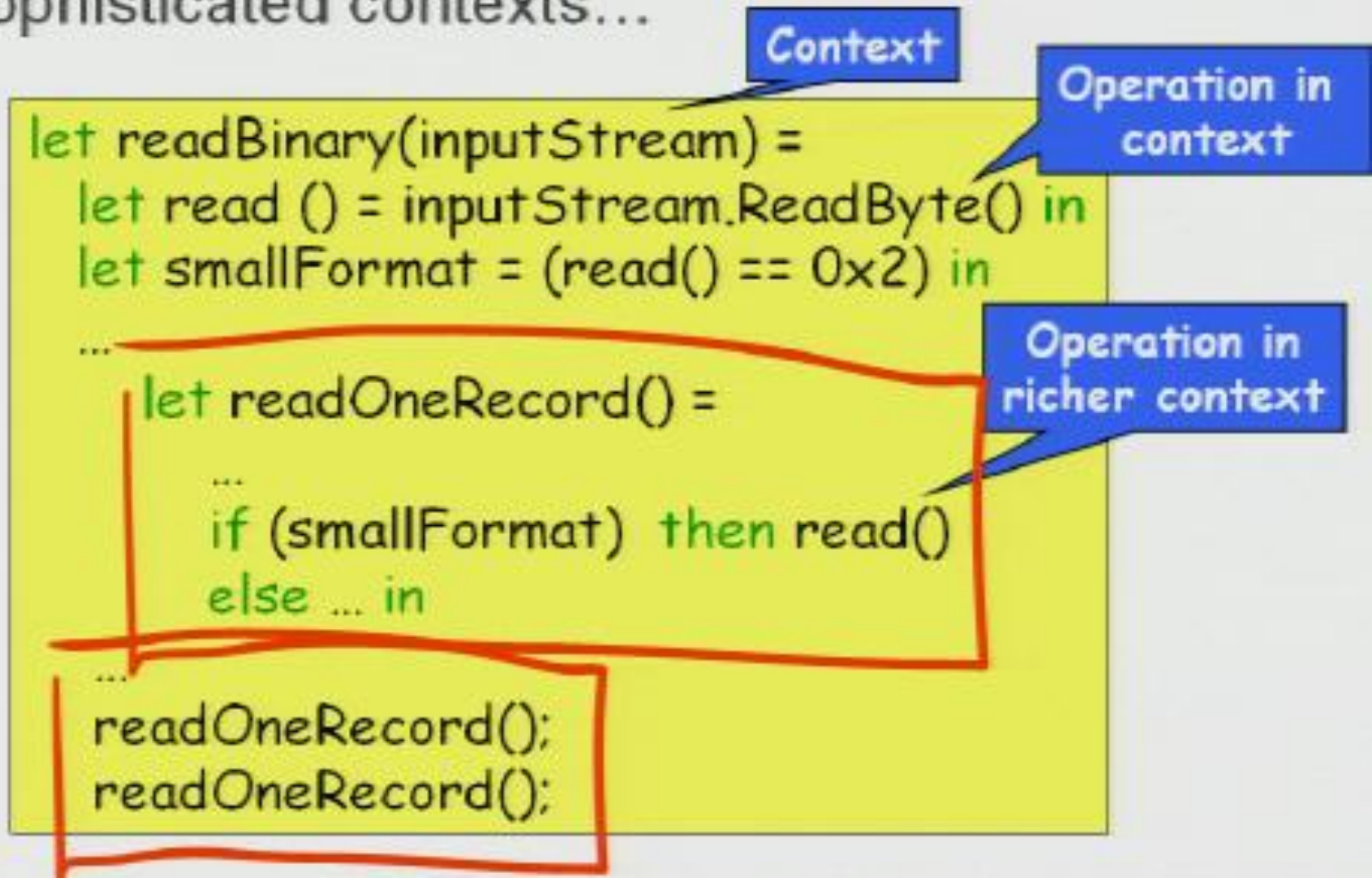
Orthogonal & Unified Constructs

- "let" + "capture": sophisticated operations in sophisticated contexts…



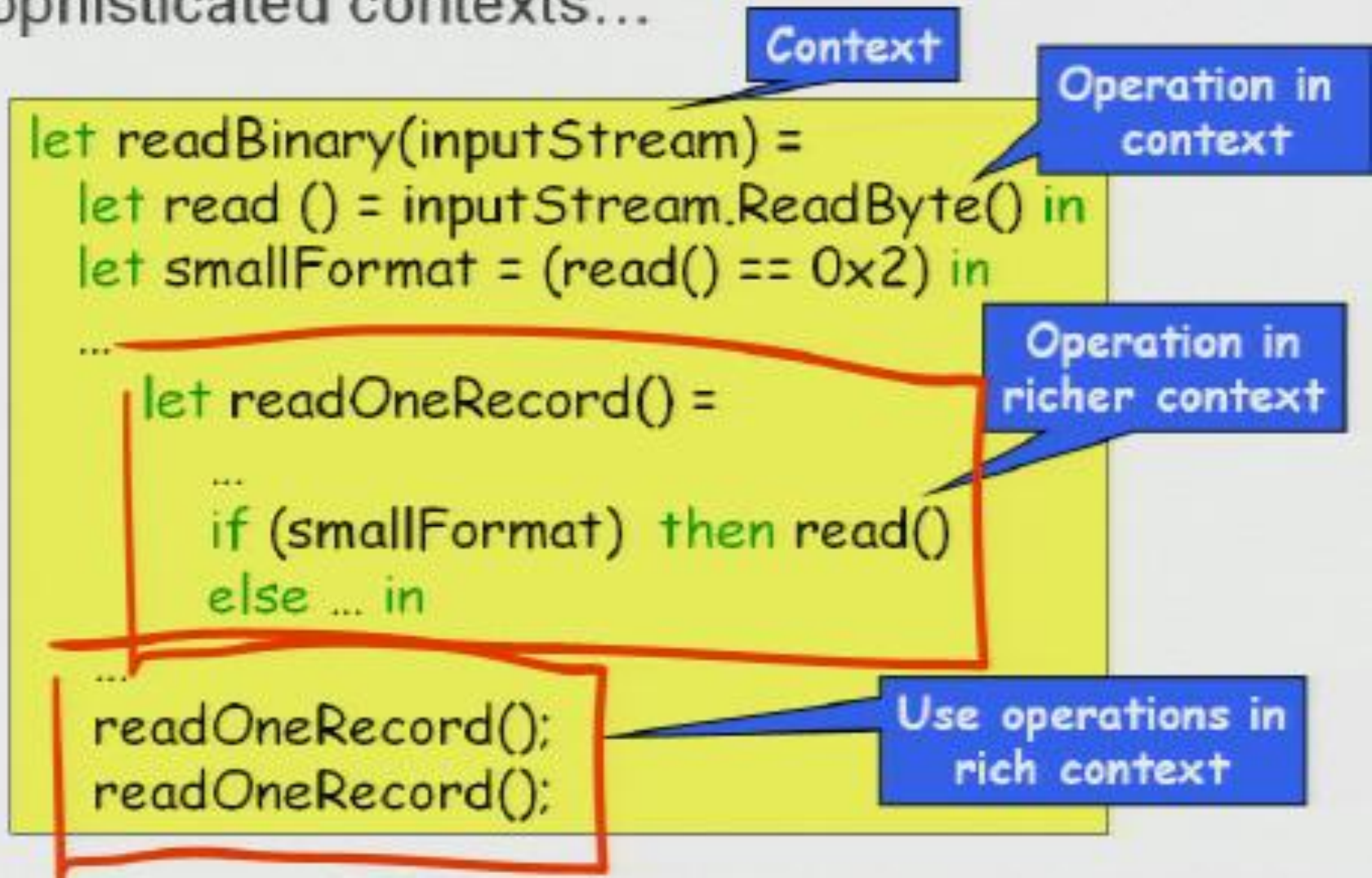
Orthogonal & Unified Constructs

- "let" + "capture": sophisticated operations in sophisticated contexts…



Orthogonal & Unified Constructs

- "let" + "capture": sophisticated operations in sophisticated contexts…



Orthogonal & Unified Constructs

- Immutability the norm…

Orthogonal & Unified Constructs

- Immutability the norm…

```
let data = 3

do data := 4
```

Orthogonal & Unified Constructs

- Immutability the norm…

```
let data = 3

do data := 4
```

```
type data =
  { Name: string;
    Size: int;
    Items: Map<string,string> }

let NewData(name,items) =
  { Name=name;
    Size=Map.size items;
    Items=items }
```

Orthogonal & Unified Constructs

→ Immutability the norm…

```
let data = 3

do data := 4
```

Simple values may not be changed

All F# data is immutable by default

```
type data =
  { Name: string;
    Size: int;
    Items: Map<string,string> }

let NewData(name,items) =
  { Name=name;
    Size=Map.size items;
    Items=items }
```

In **praise** of **immutability**

- Immutable objects can be relied on
- Immutable objects can transfer between threads
- Immutable objects can be aliased safely
- Immutable objects can be optimized

Orthogonal & Unified Constructs

- F# is a functional language
- F# is an imperative language
- **F# is a mixed functional/imperative language**
- F# is not Haskell. Loops encouraged.

Orthogonal & Unified Constructs

- Function values simplify and unify
  - …iteration

Orthogonal & Unified Constructs

→ Function values simplify and unify

…iteration

```
val List.map: ('a → 'b) → list<'a> → list<'b>
val List.foreach: list<'a> → ('a -> unit) → unit
val IEnumerable.map: ('a → 'b) → IEnumerable<'a> → IEnumerable<'b>
val IEnumerable.foreach: IEnumerable<'a> → ('a -> unit) → unit
```

Orthogonal & Unified Constructs

- Function values simplify and unify
  - ...iteration

```
val List.map: ('a → 'b) → list<'a> → list<'b>
val List.foreach: list<'a> → ('a -> unit) → unit
val IEnumerable.map: ('a → 'b) → IEnumerable<'a> → IEnumerable<'b>
val IEnumerable.foreach: IEnumerable<'a> → ('a -> unit) → unit
```

Imperative and functional versions encouraged

Orthogonal & Unified Constructs

→ Function values simplify and unify

…iteration

```
val List.map: ('a → 'b) → list<'a> → list<'b>
val List.foreach: list<'a> → ('a -> unit) → unit
val IEnumerable.map: ('a → 'b) → IEnumerable<'a> → IEnumerable<'b>
val IEnumerable.foreach: IEnumerable<'a> → ('a -> unit) → unit
```

Imperative and functional versions encouraged

Structural and Logical transformations made easy

Orthogonal & Unified Constructs

→ Function values simplify and unify

…extension

```
type UnaryOperator =
  { evaluate: (float → float);       // evaulation function
    differentiate: (expr → expr); // symbolic differentiation function
    name: string }                      // identity

let sinop = { evaluate=sin;
              differentiate=Cos;
              name="sin" }
```

Orthogonal & Unified Constructs

- Function values simplify and unify
  - ...extension

```
type UnaryOperator =
  { evaluate: (float → float);       // evaulation function
    differentiate: (expr → expr); // symbolic differentiation function
    name: string }                      // identity

let sinop = { evaluate=sin;
              differentiate=Cos;
              name="sin" }
```

Orthogonal & Unified Constructs

→ Function values simplify and unify

…extension

"Virtual methods" (extension points)

```
type UnaryOperator =
  { evaluate: (float → float);       // evaulation function
    differentiate: (expr → expr); // symbolic differentiation function
    name: string }                      // identity

let sinop = { evaluate=sin;
              differentiate=Cos;
              name="sin" }
```

Orthogonal & Unified Constructs

→ Function values simplify and unify

…extension

"Virtual methods" (extension points)

```
type UnaryOperator =
  { evaluate: (float → float);       // evaulation function
    differentiate: (expr → expr); // symbolic differentiation function
    name: string }                      // identity

let sinop = { evaluate=sin;
              differentiate=Cos;
              name="sin" }
```

Subclass? What subclass?

Orthogonal & Unified Constructs

→ Function values simplify and unify

…extension

"Virtual methods" (extension points)

```
type UnaryOperator =
  { evaluate: (float → float);       // evaulation function
    differentiate: (expr → expr); // symbolic differentiation function
    name: string }                      // identity

let sinop = { evaluate=sin;
              differentiate=Cos;
              name="sin" }
```

Caveat: Extensible extension is still hard

Subclass? What subclass?

Orthogonal & Unified Constructs

- Type parameters

```
Map<'a,'b>
List<'a>
Set<'a>
```

- Discriminated unions

```
type expr =
  | Sum of expr * expr
  | Prod of expr * expr
...
```

- Pattern matching
- Type inference
- Recursion (Mutually-referential objects)

Orthogonal & Unified Constructs

- Type parameters

```
Map<'a,'b>
List<'a>
Set<'a>
```

- Discriminated unions

```
type expr =
  | Sum of expr * expr
  | Prod of expr * expr
...
```

- Pattern matching

```
match expr with
  | Sum(a,b) -> ...
  | Prod(a,b) -> ... ...
```

- Type inference
- Recursion (Mutually-referential objects)

```
let rec map = ...
```

Less is More?

- Fewer concepts = Less time in class design
- No null pointers[1]
- Far fewer classes and other type definitions
- No constructors-calling-virtual-methods and other OO dangers

1. except leaking across from .NET APIs

Problems are problems

- Unmanaged resources (IDisposable)
- Exceptions
- I/O, avoiding blocking, concurrency
- API design
- Complex software is, well, complex…

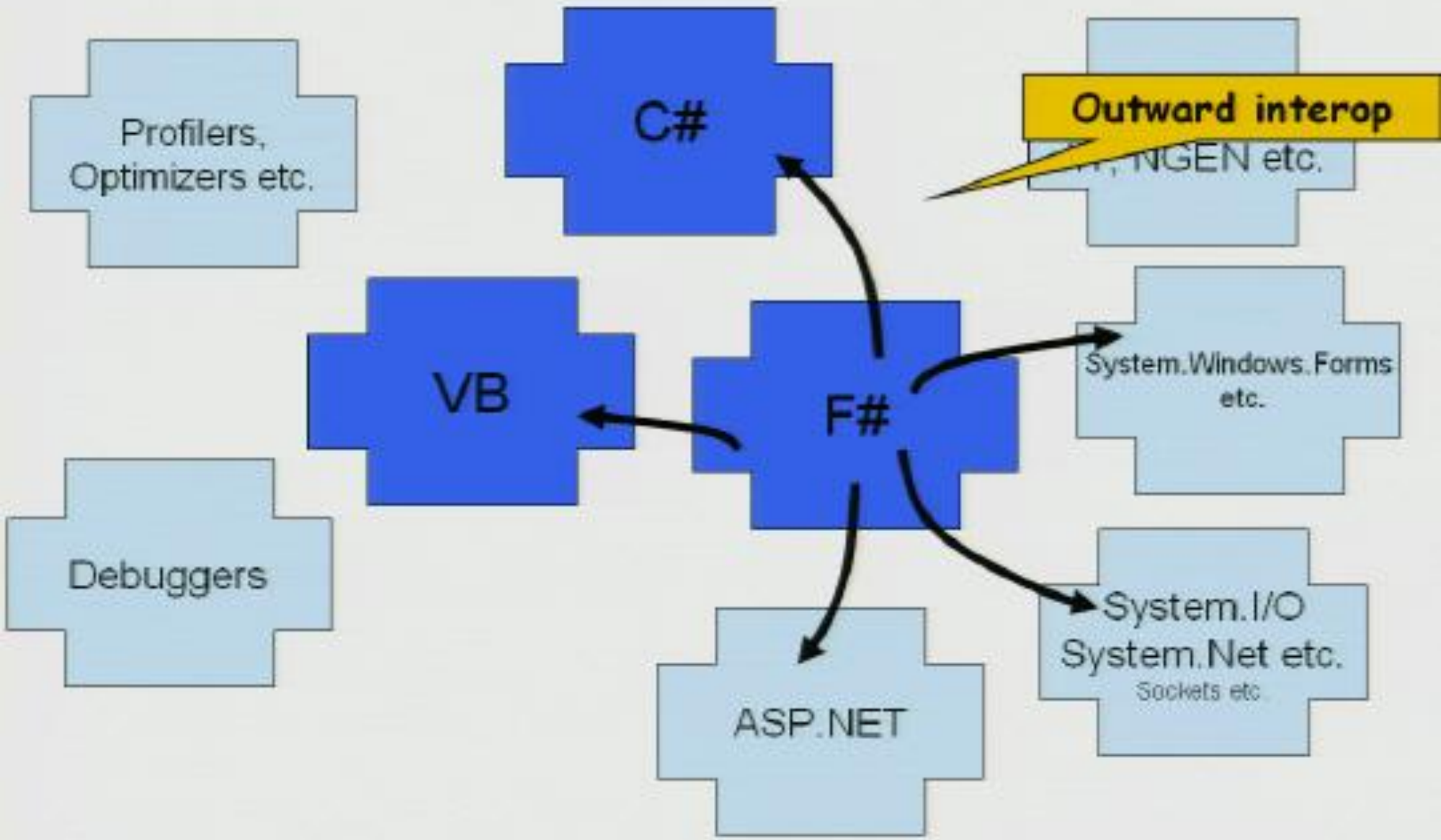
F# Observations

- F# gives you the tools to concentrate on the algorithmic complexities of symbolic processing
- F# is:
  - excellent for using .NET libraries
  - excellent for writing ML libraries
  - ok at making ML libraries usable from .NET
- So the niche seems to be for writing sophisticated applications
  - probably with some use of .NET components
  - probably with some symbolic processing
  - probably with some components written in C#

Part 2. F# Co-operates

The F# challenge: make it fit with .NET in a deeply practical way without losing the essential goodness of ML programming

Connecting F# to .NET



Outward Interop

- A very nice "." notation

```
System.Console.WriteLine("abc");

form.SetStyle(ControlStyles.AllPaintingInWmPaint,true);

form.ClientSize <- new System.Drawing.Size(292, 266);
```

- Quite similar to C#

- Delegates created using function values

Outward Interop

- Object expressions == closures for objects

```
let myForm title n =
    { new System.Windows.Forms.Form() as base
      with OnPaint(args) =
                base.OnPaint(args);
                Console.WriteLine ("OnPaint\n")
      and OnResize(args) =
                base.OnResize(args);
                Console.WriteLine ("OnResize, args = {0}\n", args)
    }
```

- Overriding only

- Orthogonal: type inference, capture, mutually recursive, nested

7 Initialization Graphs and self in Object Oriented Languages

OO languages use the value self for self referential access. One of the recurring problems in these languages is the potential for unsoundnesses that arise from the use of the object during initialization. For example, calling a virtual method in the middle of a constructor can lead to many problems. Good design dictates that no use of self be made until all fields are known to have been initialized to an appropriate value. The complexities of specifying initialization-soundness conditions for constructors in OO and OO-bytecode languages have even caused a number of security bugs in the virtual machine verifiers [12, 4].

An object's constructor forms a recursive initialization scope somewhat similar to an initialization graph with one node. Indeed, initialization graphs let you encode self references in methods without the need for a self keyword in the language at all. For example, consider the following encoding of an object as an ML record.[10]

```
type object = { getName: () -> string;
                lengthOfName: () -> int }
let skObject name =
  let rec obj = { getName = λ(). name;
                  lengthOfName =  λ(). length(obj.getName()); }
  in obj
```

[10] Encodings of object systems into ML hit limitations – for example the encoding used here does not support subtyping [1]. However that is an orthogonal issue to that discussed in this paper.

Outward Interop

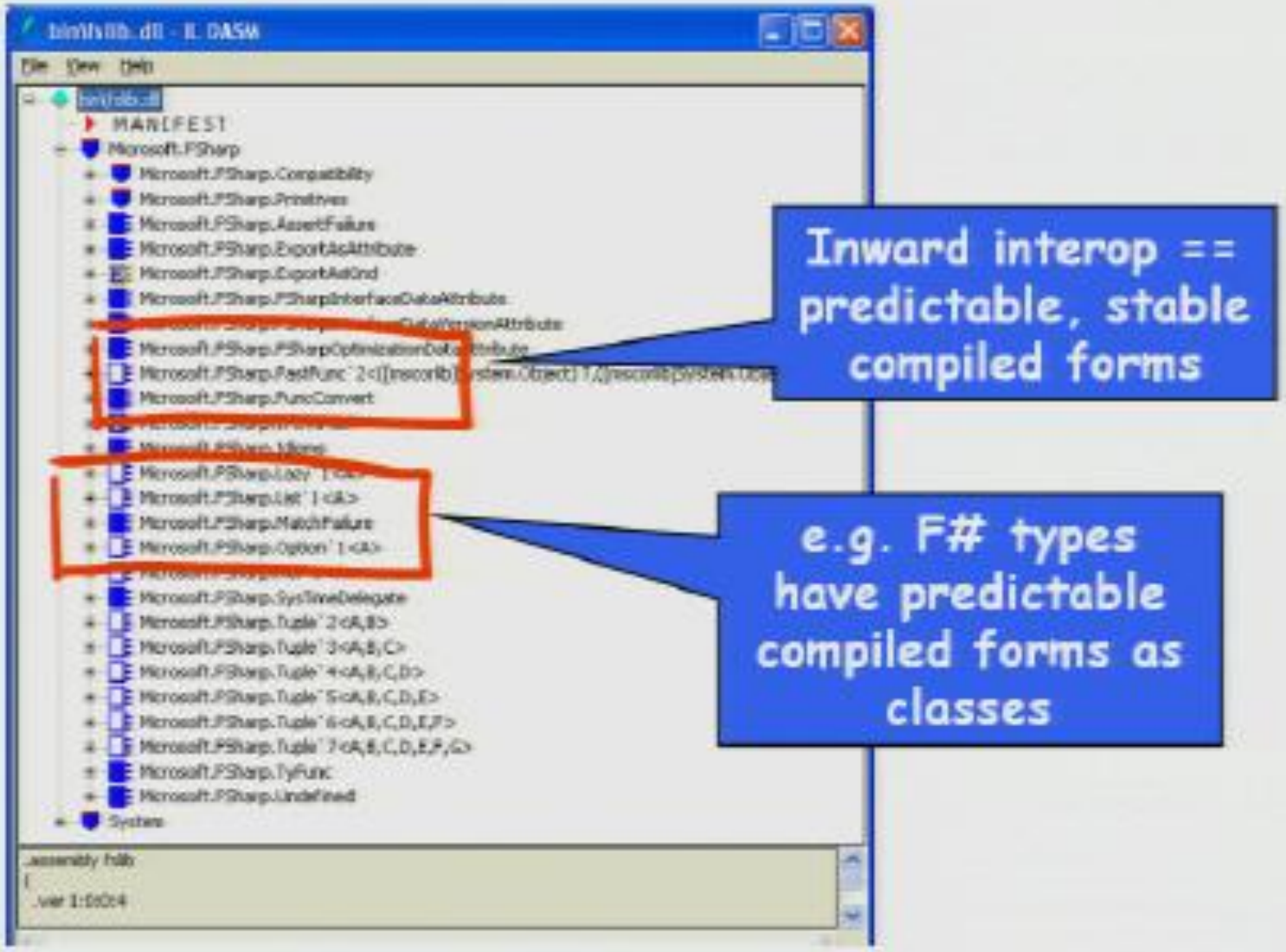
- Object expressions == closures for objects

```
let myForm title n =
    { new System.Windows.Forms.Form() as base
      with OnPaint(args) =
                base.OnPaint(args);
                Console.WriteLine ("OnPaint\n")
      and OnResize(args) =
                base.OnResize(args);
                Console.WriteLine ("OnResize, args = {0}\n", args)
    }
```

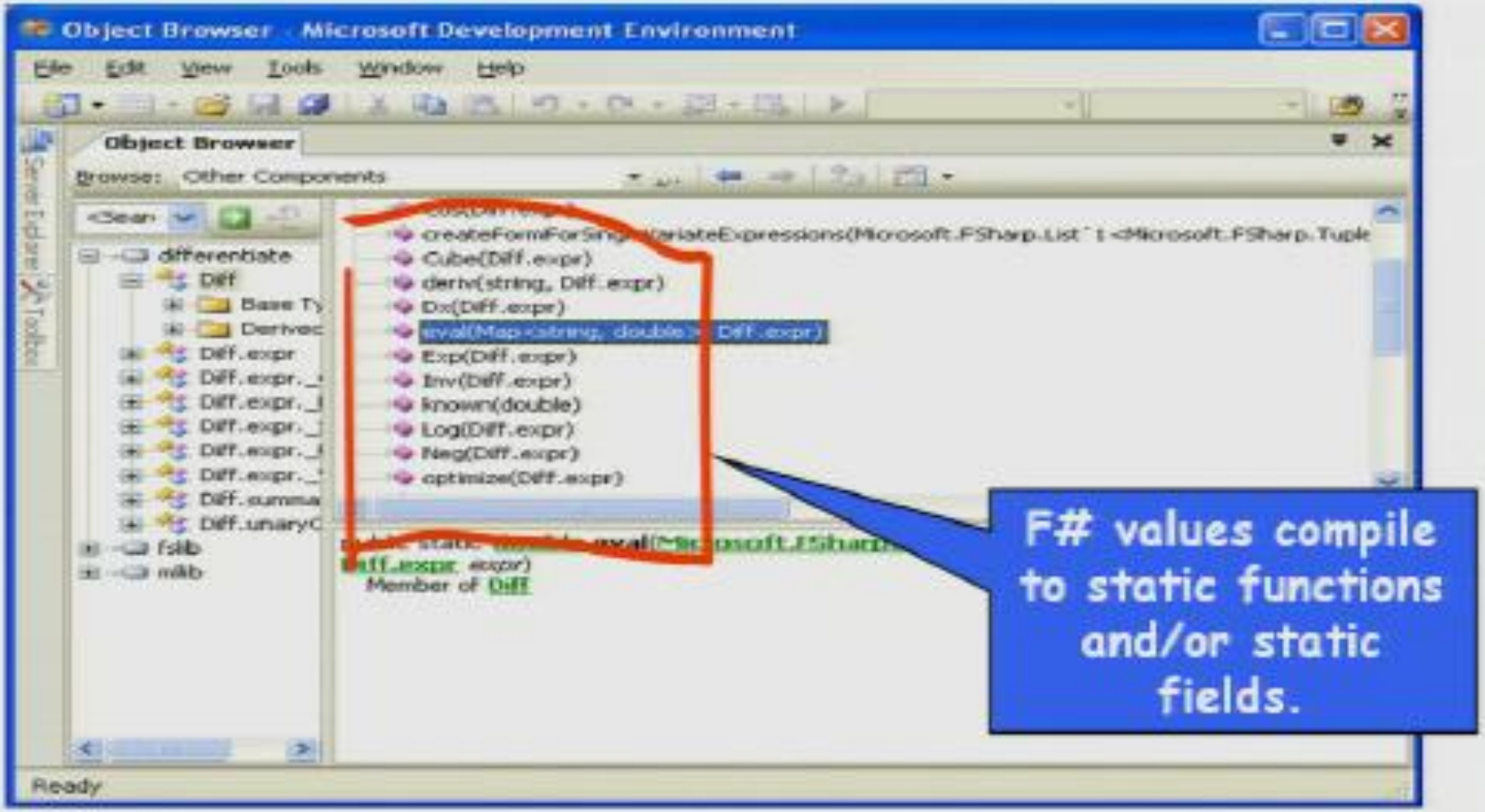
- Overriding only

- Orthogonal: type inference, capture, mutually recursive, nested

Inward Interop



Inward Interop



Inward Interop

- Inward Interop is **essential** for many reasons, e.g. **componentization**, **testing**, **profiling**
- No other ML implementation in the world has it
- Inward interop gives you full access, but is sometimes a little awkward from C#
- Room for extensions to what is there

What else does F# offer? ☺

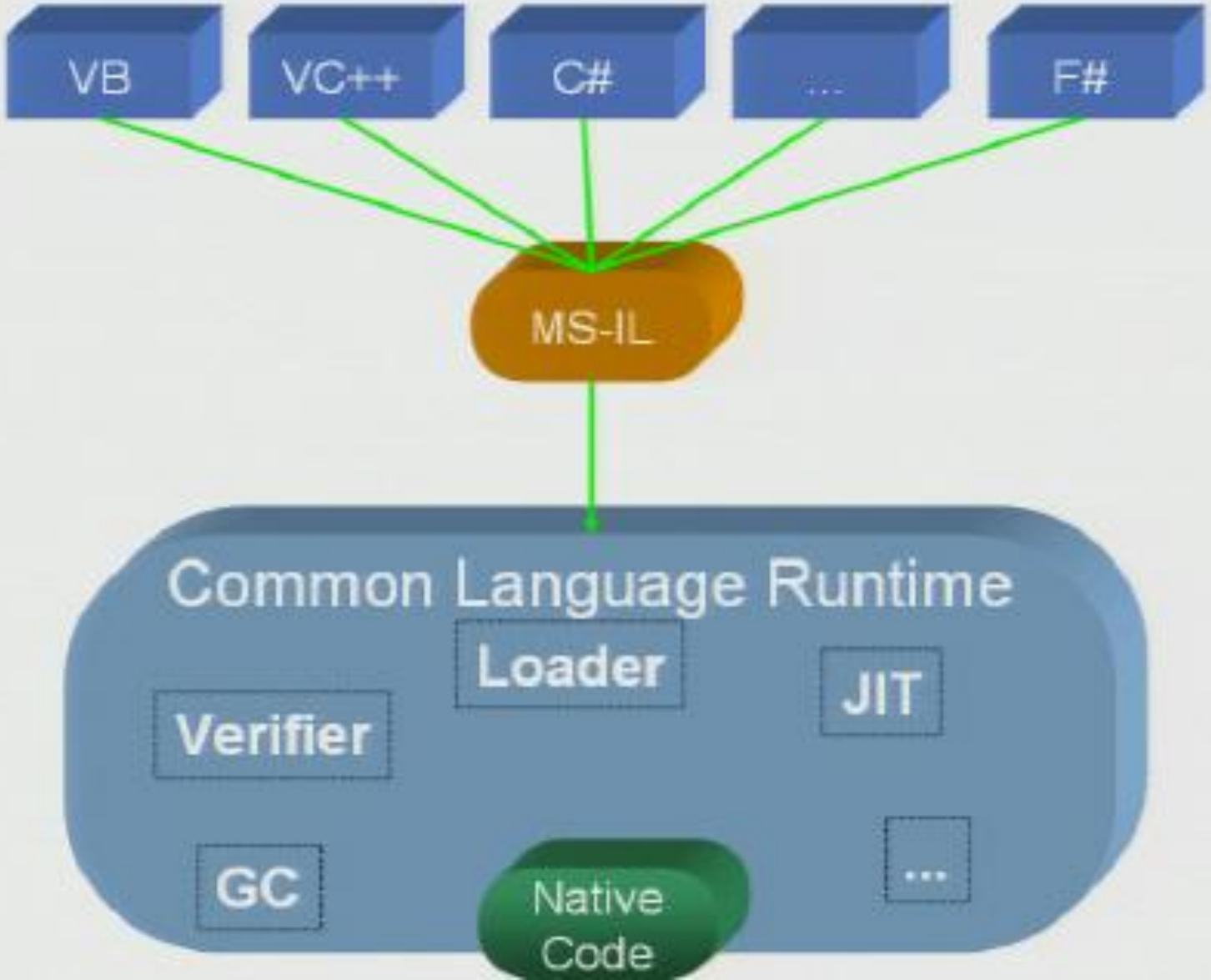
- Libraries galore
  - GUIs, Network, Speech, Graphics
- Tools galore
  - CPU Profiling, Memory Profiling, Debugging, Visual Studio integration
- C# next door
  - Fantastic interop with C and COM
- Components
  - Can build versionable, binary compatible DLLs
- Multi-threading that works
  - Even OCaml has problems here
- No significant runtime components
  - GC etc. is not part of the package

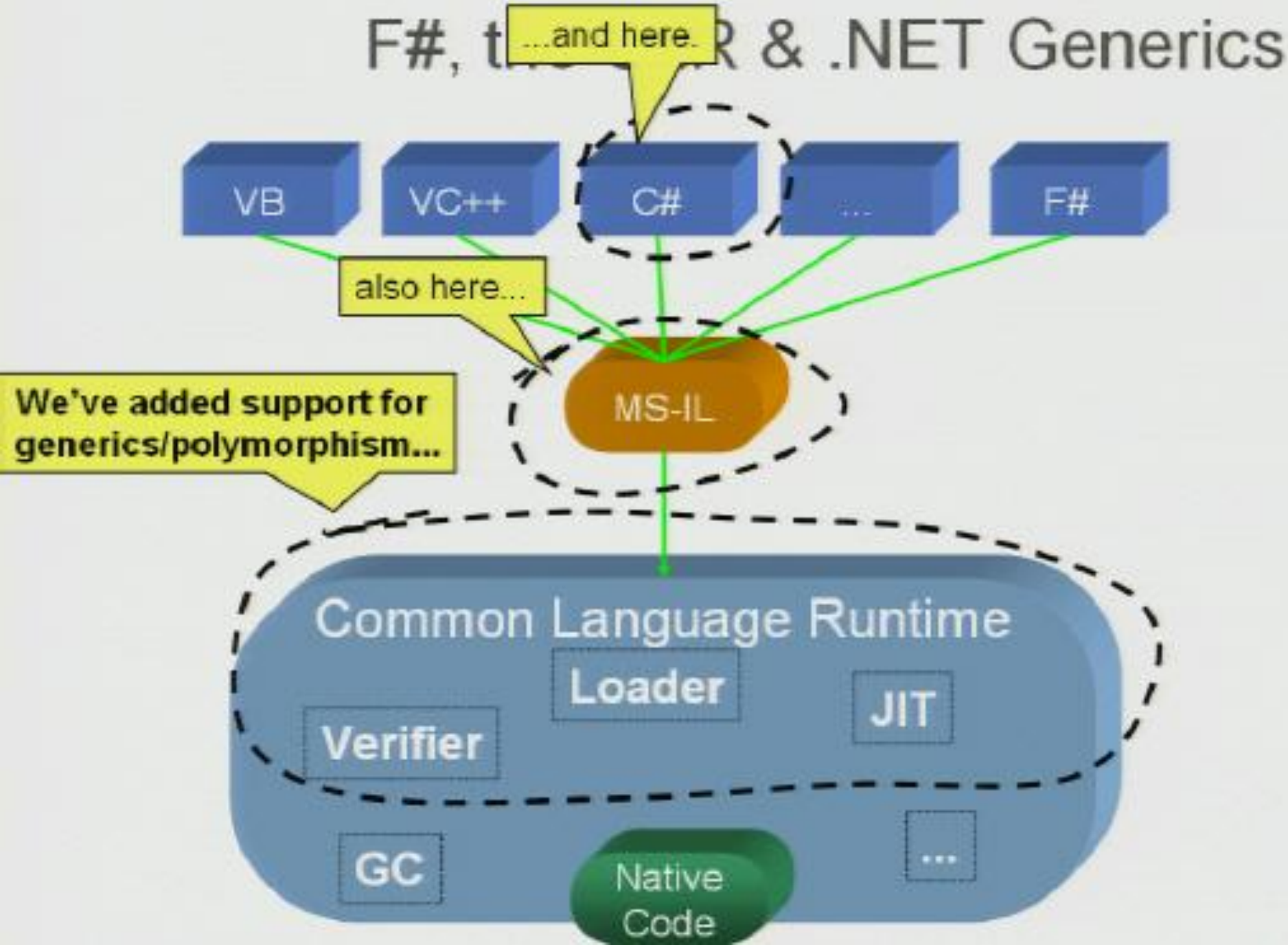
What else does F# offer? ☺

- Libraries galore
  - GUIs, Network, Speech, Graphics
- Tools galore
  - CPU Profiling, Memory Profiling, Debugging, Visual Studio integration
- C# next door
  - Fantastic interop with C and COM
- Components
  - Can build versionable, binary compatible DLLs
- Multi-threading that works
  - Even OCaml has problems here
- No significant runtime components
  - GC etc. is not part of the package

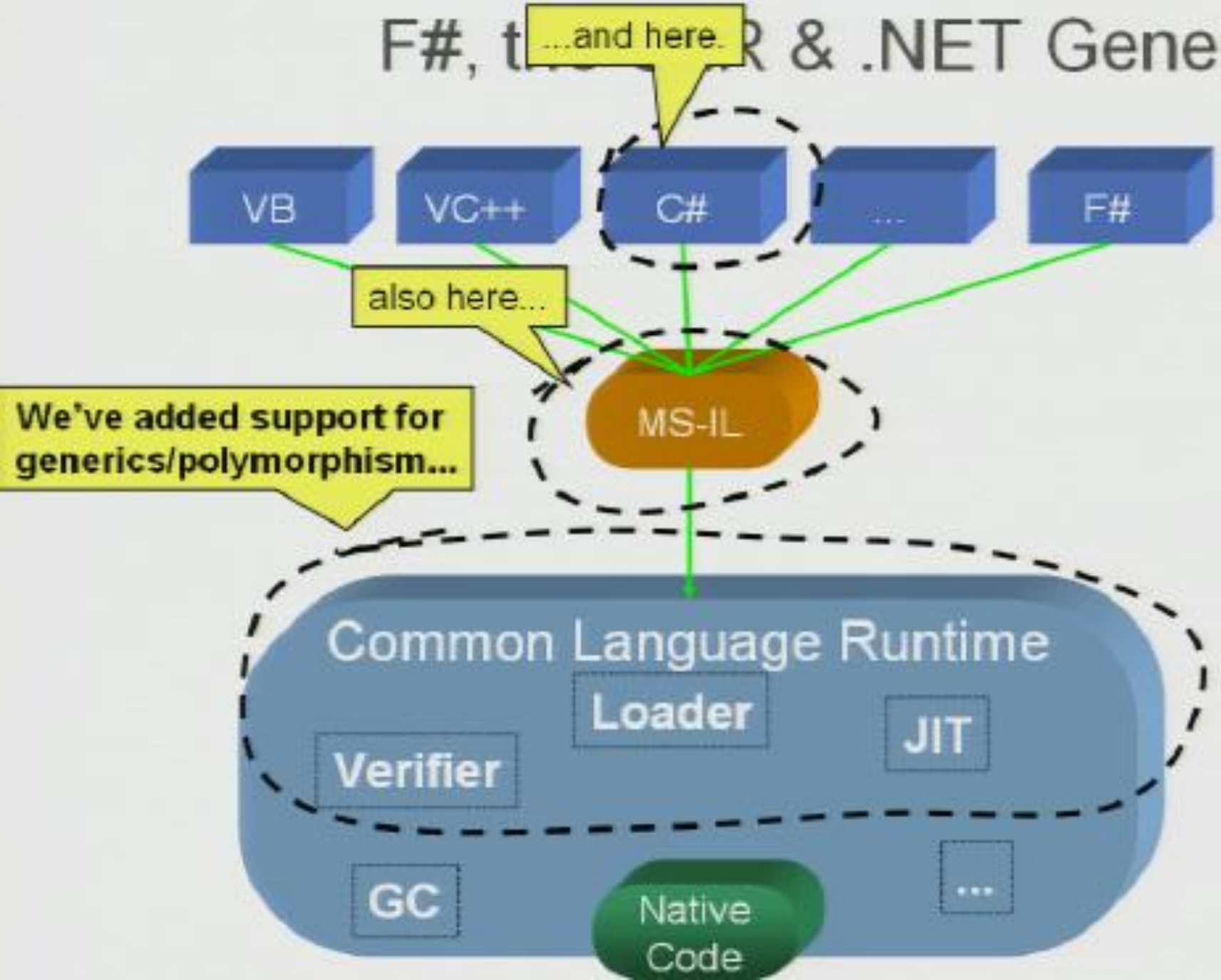
ML in .NET heaven! ☺

F#, the CLR & .NET Generics

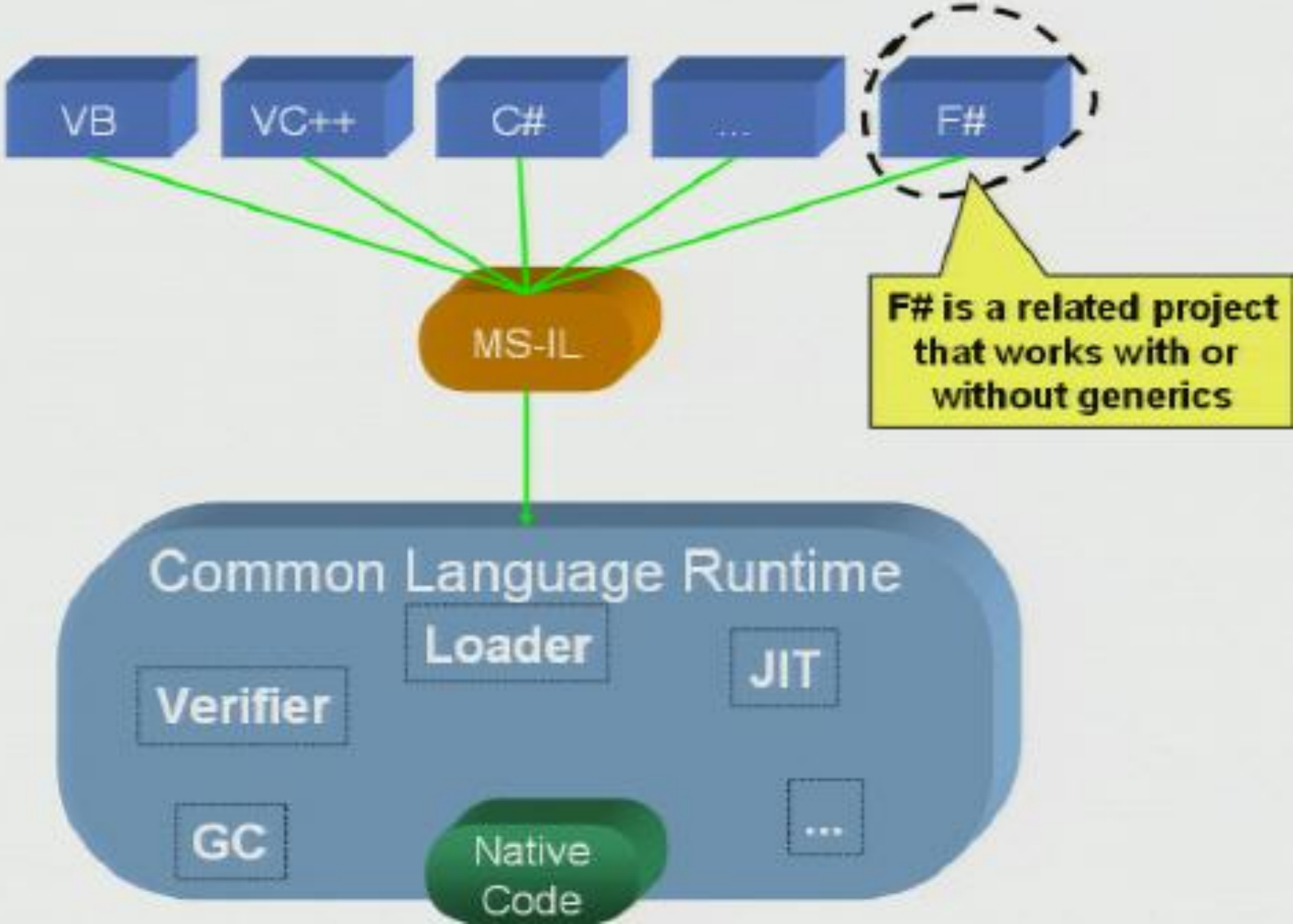



F#, t R & .NET Generics
...and here.
VB
VC++
C#
...
F#
also here...
We've added support for generics/polymorphism...
MS-IL
Common Language Runtime
Loader
JIT
Verifier
GC
Native Code
...

F#, t…R & .NET Generics



F#, the CLR & .NET Generics



F# v1.0

- Very stable core language and compiler
  - Some interesting language work will go on around the edges, e.g. ML-modules-without-ML-modules
- Being used by Zapato & SDV & many others
- ~6000 downloads/year
- VisualStudio 2005 Beta 1 integration
- ML compatibility library
- Samples etc.
- Tools: Lexer, Parser Generators

F# & Research

- Practical programming in ML still raises many interesting language design questions
- e.g. mutually recursive initialization that compensate correctly when failure occurs half-way through
- F# is a great place to conduct this research
  - easy to code examples of real-world practical programming
  - a simple clean language to extend

Questions?

http://research.microsoft.com/projects/fsharp

http://research.microsoft.com/projects/fsharp
autogroup: mailto:fsharp
\\cam-01-srv\dfsroot\projects\fcom\releases